TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JOONA LAAMANEN

TALOAUTOMAATIOJÄRJESTELMÄN LAAJENTAMINEN

Diplomityö

Tarkastaja: Prof. Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunnan tiedekuntaneuvoston
kokouksessa 17.8.2016

TIIVISTELMÄ

JOONA LAAMANEN: Taloautomaatiojärjestelmän laajentaminen
Tampereen teknillinen yliopisto
Diplomityö, 52 sivua
Tammikuu 2017
Tietotekniikan koulutusohjelma
Pääaine: Ohjelmistotuotanto
Tarkastajat: Prof. Tommi Mikkonen
Avainsanat: taloautomaatio, kotiautomaatio, älytalo, älykoti

Taloautomaatiolla tarkoitetaan tapaa automatisoida talon erilaisia toimintoja elektroniikan avulla. Arkikielessä taloautomaatiota käyttävistä taloista käytetään usein nimitystä *"älykoti"* tai *"älytalo"*. Taloautomaation avulla voidaan toteuttaa talon toiminnoille älykästä ohjausta, kuten esimerkiksi lämmityksen ja ilmastoinnin ohjausta lämpö- ja kosteusanturien lukemien mukaisesti. Taloautomaatiojärjestelmille on tyypillistä myös toimintojen keskitetty ohjaus ohjauspäätteen tai älylaitteen avulla.

Tässä diplomityössä kehitetään laajennus olemassa olevaan taloautomaatiojärjestelmään. Järjestelmän nykyiset ohjauspäätteet ovat hitaita, vaikeita käyttää sekä vaikeasti laajennettavia. Laajennus korvaa vanhat ohjauspäätteet ja tekee järjestelmästä nopeamman ja helpomman käyttää.

Laajennus päätettiin toteuttaa web-teknologioilla, koska jatkokehityksessä järjestelmä on tarkoitus laajentaa pilvipalveluksi. Palvelinohjelmisto tarjoaisi tätä pilvipalvelua varten valmiin rajapinnan ja käyttöliittymäkoodikin olisi uudelleenkäytettävää. Web-ohjelma esitetään ohjauspäätteellä yksinkertaisen Qt-ohjelman avulla. Arkkitehtuurin pohjana käytettiin MVC-mallia, jota valitut sovelluskehykset tukevat. Sovelluslogiikkaa jaettiin myös toimintakohtaisesti palveluihin modulaarisuuden edesauttamiseksi.

Ohjauspäätteen käyttö on laajennuksen ansiosta huomattavasti nopeampaa ja helpompaa, joten työ onnistui tavoitteessaan. Lisäksi asiakas on lopputulokseen tyytyväinen, vaikka toimituksen aikataulu viivästyikin hieman alkuperäisestä. Suurimmat haasteet olivat projektin laajuuden vaihtelu sekä kokemuksen puute valituista teknologioista.

ABSTRACT

JOONA LAAMANEN: Extending a house automation system
Tampere University of Technology
Master's Thesis, 52 pages
January 2017
Master's Degree Programme in Information Technology
Major: Software engineering
Examiner: Prof. Tommi Mikkonen
Keywords: house automation, home automation, smart house, smart home

House automation is a way to automate different house functionalities by means of electronics. In spoken language houses using house automation are often referred to as "smart homes" or "smart houses". With the help of house automation, it is possible to implement smart controlling for house functionalities, including heating and ventilation control in response to thermometer and humidity sensor readings. It is also typical for house automation systems to centralize controls to a control panel or smart devices.

In this thesis, an extension for an existing home automation system is developed. The current control panels of the system are slow, difficult to use and difficult to expand. The extension will replace the previous control panels and it will improve the efficiency and usability of the system.

The extension was decided to be implemented with web technologies, because the client has intentions of expanding the system into a cloud service in further development. The server software would offer a ready-to-use interface for the cloud service, and user interface code would also be reusable. On the control panel, the web program is presented as a simple Qt program. MVC model was used as a basis for architecture design, because it is used by chosen web frameworks. Application logic was also divided to services by function to improve modularity of the software.

Using the control panel is considerably faster and easier because of the extension, thus the result is successful. The client was also satisfied with the result, although the delivery was delayed from the original schedule. The biggest challenges were the alternating project scope and lack of competence with the chosen technologies.

ALKUSANAT

Tämä diplomityö perustuu syksyllä 2015 alkaneeseen ja alkuvuodesta 2016 päättyneeseen projektiin.

Haluaisin kiittää työnantajaani Ciniaa sekä asiakastamme Control Intelligenceä mahdollisuudesta tehdä diplomityö projektiin liittyen. Kiitos Kyösti Rannolle ohjauksesta sekä yhteistyöstä projektin kehityksessä ja suunnittelussa, sekä kiitos Petri Nukariselle projektinhallinnasta. Kiitos myös professori Tommi Mikkoselle ohjauksesta ja vinkeistä työn kirjoittamiseen liittyen.

Kiitos myös Sinille ja Olavi-pojalle tuesta, kärsivällisyydestä ja rakkaudesta!

Nokia, 22.12.2016

Joona Laamanen

SISÄLLYS

1. Johdanto 1

2. Taloautomaatio 2

2.1 Historiaa 2

2.2 Tekninen tausta 4

2.3 Yleiset haasteet 5

3. Järjestelmän pohja 7

3.1 Jakokeskus 7

3.2 Ohjausyksiköt 7

3.3 Kytkimet ja anturit 9

3.4 Vanhat ohjauspäätteet 10

3.5 Uusi ohjauspääte 11

4. Järjestelmän laajennus 12

4.1 Tavoitteet ja kriteerit 12

4.2 Vaihtoehto 1: alustariippumattomat sovelluskehykset 13

4.3 Vaihtoehto 2: web-teknologiat 14

4.4 Vertailu ja valinta 17

5. Laajennuksen toteutus 21

5.1 Toteutettavat toiminnallisuudet 21

5.2 Suunnittelu 22

5.3 Kehitys 26

5.4 Käyttöliittymän kuvaus 28

5.5 Frontend 32

5.6 Backend 36

6. Arviointi 44

6.1 Tavoitteiden toteutuminen 44

6.2 Ongelmat ja haasteet 46
6.3 Kehityskokemukset 47
6.4 Jatkokehitysideoita 48
7. Yhteenveto 50
Lähteet 51

KUVALUETTELO

2.1 Yleinen taloautomaatioarkkitehtuurin malli. 4

3.1 Kuva taloautomaatiojärjestelmän arkkitehtuurista. 8

3.2 Kuva järjestelmän sähkökaapista. 9

3.3 Kuva vanhoista ohjauspäätteistä. 10

3.4 Kuva uudesta Chipsee-ohjauspäätteestä edestä ja takaa. 11

4.1 Asiakas-palvelin-malli. 15

4.2 MEAN-pino. 16

5.1 Kuva MVC-arkkitehtuurista. 23

5.2 Yksinkertaistettu kuva laajennuksen ohjelmistoarkkitehtuurista. . . . 24

5.3 Kuva laajennuksen järjestelmäarkkitehtuurista. 25

5.4 Näytönkaappaus valaistusnäkymästä. 26

5.5 CAN-proxyn toiminta kehitysympäristössä. 28

5.6 Frontend-arkkitehtuurikaavio. 33

5.7 Backend-arkkitehtuurikaavio. 36

1. JOHDANTO

Teknologia kuuluu yhä tiiviimmin ihmisten arkielämään. Yhä useampaa kodinkonetta ohjaa jonkinlainen mikropiiri, minkä avulla laitteista voidaan tehdä älykkäämpiä. Elektroniikan avulla on nykyään realismia toteuttaa kokonaisen kodin kattavia laitteiden verkkoja ja esimerkiksi ohjata automaattisesti ilmastointia lämpötila- ja kosteusanturien lukemien mukaisesti, tai vaikka ohjata koko talon valaistusta keskitetysti ohjauspäätteeltä tai omalta älylaitteelta. Tätä kutsutaan *taloautomaatioksi*.

Tässä diplomityössä kehitetään laajennus olemassa olevaan, CAN-väylään perustuvaan taloautomaatiojärjestelmään. Järjestelmällä voidaan ohjata muun muassa talon LED-valaistusta, lämmitystä sekä murtohälytintä. Järjestelmää käytetään nykyään ohjauspäätteillä, jotka ovat vaikeita ja hitaita käyttää. Laajennus korvaa järjestelmän edelliset ohjauspäätteet sekä nopeuttaa ja helpottaa järjestelmän käyttöä.

Luvussa 2 esitellään taloautomaatiota ja sen historiaa ja arkkitehtuureita yleisellä tasolla sekä käydään läpi yleisiä taloautomaatiojärjestelmien haasteita. Tämän jälkeen luvussa 3 esitellään asiakkaan toimittaman taloautomaatiojärjestelmän pohja, joka on laajennuksen kohteena. Järjestelmän pohjasta esitellään arkkitehtuuri ja komponentit, sekä uusi ohjauspääte, joka on tilattu laajennuksen toteutusta varten.

Luvussa 4 esitellään järjestelmän laajennuksen tavoitteet, joiden perusteella tehdään toteutukselle teknologiavertailua. Vertailun tuloksena esitellään valittu toteutusteknologia. Luvussa 5 esitellään laajennuksen toteutus käymällä läpi toteutettavat toiminnallisuudet sekä arkkitehtuurin suunnittelua. Myös kehitysympäristöä esitellään, jonka jälkeen esitellään frontend- ja backend-toteutuksia yksityiskohtaisemmin arkkitehtuuritasolla.

Luvussa 6 laajennuksen toteutus arvioidaan, kuinka hyvin se täytti projektin alussa asetetut tavoitteet. Myös projektin aikana esiintyneet ongelmat, kehityskokemukset sekä jatkokehitysideat esitellään. Lopuksi luvussa 7 työn lopputulos käydään tiivistetysti läpi yhteenvedossa.

2. TALOAUTOMAATIO

Taloautomaatiolla tarkoitetaan tapaa automatisoida talon erilaisia toimintoja elektroniikan avulla. Arkikielessä taloautomaatiota käyttävistä taloista käytetään usein nimitystä *"älykoti"* tai *"älytalo"*. Taloautomaation avulla voidaan toteuttaa talon toiminnoille älykästä ohjausta, kuten esimerkiksi lämmityksen ja ilmastoinnin ohjausta lämpö- ja kosteusanturien lukemien mukaisesti. Taloautomaatio mahdollistaa myös toimintojen ohjaamisen keskittämisen ohjauspäätteestä tai kaukosäätimestä käytettäväksi. Talo voidaan määritellä älykkääksi, jos siinä on käytössä tietoteknisiä toimintoja vastaamaan asukkaiden erilaisiin tarpeisiin.

Tämä luku perustuu Richard Harperin kirjoittamaan artikkelikokoelmaan *Inside the Smart Home* [10], jollei toisin mainita.

2.1 Historiaa

Taloautomaatio on ollut jo kauan aihe tieteisfiktiossa, mutta ensimmäisiä kertoja sitä toteutettiin käytännössä 1960-luvulla, jolloin ensimmäisiä "langallisia koteja" tehtiin harrastajien toimesta. Ensimmäisen kerran termiä älykoti käytettiin kuitenkin vasta vuonna 1984, kun kodinelektroniikka alkoi yleistyä ja teknologian interaktiivisuus alkoi edistyä. Interaktiivisuus mahdollisti käyttöliittymien suunnittelun ja toteuttamisen kuluttajille sopiviksi.

Älykodit ovat yleistyivät kuitenkin 1990-luvun markkinoilla hitaasti. Tämä johtuu muun muassa siitä, että olemassa oleviin taloihin on kallista rakentaa älykodin vaatimaa verkkoa. Yhteistä protokollaa talon laitteiden ohjaukseen ei myöskään ollut olemassa, joten älykotien ohjausjärjestelmät keskittyivät yksinkertaisiin päällekytkentöihin. Käytettävyyteen ei myöskään panostettu, koska käytettävyyden arviointi oli haastavaa johtuen vaihtelevasta käyttäjäkunnasta.

Suurelle yleisölle älykodista tuli lopullisesti todellisuutta, kun useat aikakauslehdet

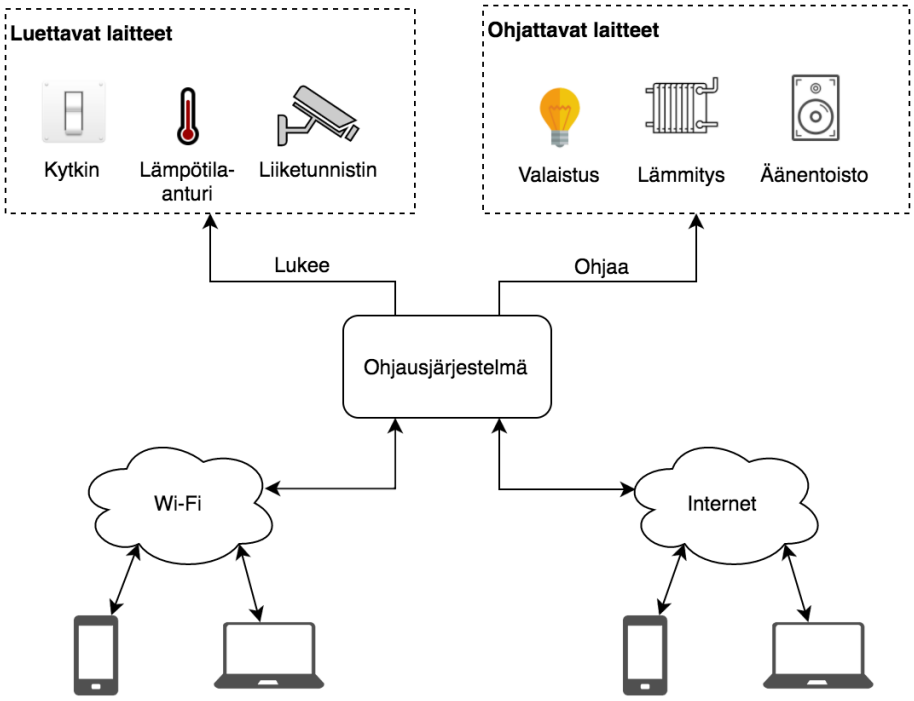
alkoivat julkaista artikkeleita älykotiin liittyen. Lisäksi BBC esitti vuonna 1999 dokumenttisarjan *Dream House*, joka kuvasi älykodissa asuvaa perhettä kuuden viikon ajan. Vaikkakin tietoisuus älykodeista kasvoi, kovinkaan moni ei halunnut omasta kodistaan älykästä. Tähän liittyi tiiviisti pelko talon liiallisesta valvonnasta ja tekoälyn kehittymisestä, mitä kuvastaa hyvin lausahdus "kuinka älykäs sänkysi pitää olla, ennen kuin pelkäät mennä nukkumaan?" [7].

2000-luvulla yleistyneiden sulautettujen kehitysalustojen ansiosta älykotien kehittäminen tuli kaikkien saataville. Tällaisia kehitysalustoja ovat mm. Arduino ja Raspberry Pi. Näiden alustojen ja niiden tarjoamien yhteisöjen avun myötä kuka tahansa pystyy edullisesti ja helposti toteuttamaan joitakin älykodin toiminnallisuuksia. Tähän perehtyminen kuitenkin vaatii käyttäjältä aikaa, vaivaa ja jonkin verran tietoteknistä osaamista.

Älypuhelinten ja muiden kosketusnäytöllisten laitteiden suuren suosion kasvun myötä 2010-luku on ollut eräänlainen taitekohta, joka määrittelee pitkälti vaatimukset myös älykodin käytettävyydelle. Älypuhelimille ja tableteille on julkaistu useita eri ohjelmia ja laitteita, jotka mahdollistavat älykodin ohjauksen joko pilvipalvelimen tai paikallisen palvelimen kautta. Nämä kuluttajan itse asennettavissa olevat älykotitoiminnot vaativat kuitenkin ohjaukseen soveltuvat laitteet, kuten lamppujen polttimot ja pistorasiaohjaimet, jotka aiheuttavat lisäkustannuksia.

Pilvipalvelimeen perustuvat älykotiohjaukset ovat kuitenkin nostaneet useita kysymyksiä yksityisyydensuojaan, tietoturvaan ja palvelun toimivuuden jatkuvuuteen liittyen. Esimerkiksi eräs älykotituotteen tarjoaja ilmoitti lopettavansa pilvipalvelun ylläpidon laitteiden takuun umpeutumiseen vedoten, mikä tekee tuotteeseen kuuluvista laitteista hyödyttömiä [6].

Nykypäivänä eräs nopeimmin yleistyvästä taloautomaatiojärjestelmistä on Philips Hue -järjestelmä. Huessa korostuu asennuksen yksinkertaisuus, sillä älylaitteilla säädettävän valaistuksen saa aikaan asentamalla valaisimeen "älykkään" Hue-lamppuun sekä langattoman siltaimen avulla. Hue-lamppu käyttää lyhytkantoista langatonta ZigBee-verkkoa ja yhdistyy älylaitteiden saatavilla olevaan verkkoon siltaimen kautta [12].



Kuva 2.1 *Yleinen taloautomaatioarkkitehtuurin malli.*

2.2 Tekninen tausta

Taloautomaatiojärjestelmille on tyypillistä, että ohjaus keskitetään ohjausjärjestelmään, joka ohjaa ja lukee järjestelmän laitteita ja antureita yksinkertaisilla viesteillä [8]. Ohjausjärjestelmä voidaan myös hajauttaa useaan eri yksikköön riippuen talon koosta ja johdotuksen tarpeista, jolloin yksittäinen yksikkö ohjaa ja lukee vain siihen kytkettyjä laitteita, mutta välittää tiedon myös muille yksiköille. Tällöin laitteiden ohjauslogiikka voidaan joko keskittää yhteen yksikköön tai toteuttaa se kaikkiin yksiköihin, jolloin yksiköt voisivat toimia itsenäisemmin ja järjestelmä olisi vikasietoisempi.

Ohjausjärjestelmä voi tarjota Wi-Fi -rajapinnan, minkä avulla samassa verkossa olevilla älylaitteilla ja tietokoneilla voidaan myös ohjata järjestelmää. Ohjausjärjestelmä voi tarjota myös internet-rajapinnan etäkäyttöä varten. Yleinen malli taloautomaatiojärjestelmän arkkitehtuurista on esitetty kuvassa 2.1.

Ohjausjärjestelmän ja laitteiden välinen kommunikointi voidaan toteuttaa fyysisesti johdoilla, mutta langattomuus on nykyään yleistä sen pienten kustannusten ja helpon asennuksen takia. Langatonta kommunikointia voidaan toteuttaa langattoman

ZigBeen avulla. ZigBee-verkot ovat lyhytkantoisia, mutta ZigBee-verkkoon kytketyt laitteet muodostavat vertaisverkon kaltaisen topologian, eli verkon kattavuutta voidaan laajentaa ketjuttamalla sitä laitteelta toiselle [2]. Lisäksi ZigBee on energiatehokas tiedonsiirtotapa verrattuna esimerkiksi Wi-Fiin, mutta tukee vain pieniä datansiirtonopeuksia [8].

ZigBeen lisäksi langattomassa tiedonsiirrossa voidaan käyttää myös Wi-Fiä tai Bluetoothia, mikä on tarpeen varsinkin viihde-elektroniikkalaitteisiin kytkeytyessä. Järjestelmällä voidaan ohjata esimerkiksi kytkinten tai älypuhelimen avulla äänentoistojärjestelmää, jos siinä on avoin ohjausrajapinta [8].

Fyysisesti johdotettu järjestelmä on asennuskustannusten ja huollon takia kalliimpi vaihtoehto, mutta luotettavampi sekä tietoturvan että toimintavarmuuden kannalta. Eräs vaihtoehto toteuttaa johdotus on käyttää *I2C*-väylää. Väylä perustuu parikaapeliin, jossa on data- ja kellojohtimet. Mikä tahansa laite voi aloittaa viestin lähetyksen ajamalla data- ja kelloulostuloja tietyllä tavalla, jolloin muut väylään kytketyt laitteet alkavat vastaanottaa viestiä. I2C on noodikeskeinen väylä, eli toisin sanoen sen viesteillä on aina määritelty vastaanottavan laitteen osoite. Vastaanottava laite kuittaa viestin vastaanotetuksi ACK-sanomalla tai virheen tapahtuessa pyytää lähetystä uudelleen NACK-sanomalla [16].

Toinen väylävaihtoehto on usein autojen elektroniikassa käytetty CAN-väylä, joka on I2C:n tavoin parikaapelilla toimiva väylä. Myös CAN-väylällä kaikki laitteet voivat lähettää viestejä muille laitteille, mutta samanaikaisesti lähetettävien viestien priorisointi tapahtuu laitteiden ID-numeroiden perusteella. Suurin ero I2C-väylään on siinä, että se on viestikeskeinen, eli kaikki viestin vastaanottavat laitteet käsittelevät sen [17].

2.3 Yleiset haasteet

Yksi suurimmista haasteista taloautomaation yleistymiselle ovat järjestelmän asennuksen kustannukset. Fyysisesti laitteiden yhdistäminen johdotuksella on kallista, koska se vaatii yleensä johtojen asentamista seinän sisälle. Langattomasti asennettavat järjestelmät ovat asennuksen puolesta edullisempia, mutta langaton tiedonsiirto kuluttaa enemmän virtaa ja aiheuttaa mahdollisen tietoturvariskin, jos ulkopuolinen käyttäjä pääsee ohjaamaan järjestelmää talon ulkopuolelta.

Tietoturva on vakava riski varsinkin järjestelmissä, joita voidaan ohjata pilvipalve-

lun kautta. Järjestelmään murtautuminen ei tällöin vaatisi sitä, että hyökkääjä olisi fyysisesti taloa lähellä langattoman reitittimen kantaman sisällä, vaan pelkkä pilvipalvelun tunnusten murtaminen riittää pääsemään käsiksi järjestelmään. Tämän takia joitakin ominaisuuksia, kuten murtohälyttimen ohjausta, olisi järkevä estää varotoimena kokonaan toimimasta pilvipalvelun kautta.

Järjestelmän huolto ja ylläpito ovat haasteita varsinkin johdotetuissa järjestelmissä. Langattomiin laitteisiin voidaan toteuttaa käyttäjän itsensä suorittama laitteiden päivitys, mutta fyysisten liitäntöjen korjaamista varten palveluntarjoajan tulisi tarjota huoltopalveluita.

3. JÄRJESTELMÄN POHJA

Olemassa oleva järjestelmän pohja kehitettiin alunperin ambulanssien sisävalaistuksen ja toimintojen ohjaamista sekä vianmääritystä varten. Tämän takia järjestelmän komponentit ovat edelleen kytkettyinä toisiinsa CAN-väylän avulla, koska suurin osa nykyajan autoista tukee CAN-protokollaa.

Järjestelmä koostuu useasta eri osasta, kuten noodeista, kytkimistä, antureista ja jakokeskuksesta. Tässä luvussa käydään järjestelmän pohjan osat ja niiden toiminta läpi. Järjestelmän arkkitehtuuri on kuvattu kuvassa 3.1.

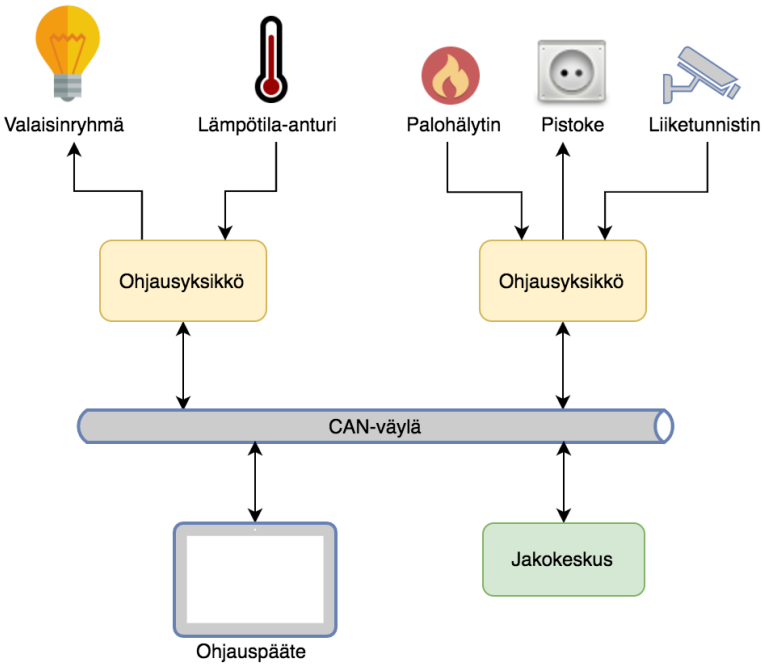
3.1 Jakokeskus

Järjestelmään kuuluvat laitteet asennetaan asiakkaiden koteihin omiin sähkökaappeihinsa. Kaappeja voi olla toisiinsa kytkettyinä useita, sekä jokaisella kaapilla on oma jakokeskuksena. Jakokeskus jakaa virtalähteestä tulevan virran muihin sitä tarvitseviin laitteisiin eli ohjausyksiköihin ja GSM-modeemiin. Virta ajaa myös ohjausyksiköiden lähtöjä, kuten LED-valaisimia. Kaappi on esitetty kuvassa 3.2.

Jakokeskuksen avulla voidaan myös päivittää kunkin siihen liitetyn ohjausyksikön ohjelmistot sekä muuttaa ohjausyksiköiden muistissa ylläpidettäviä parametreja, kuten murtohälyttimen poistumisviivettä. Tämä päivitys tapahtuu kytkemällä tietokone sarjaportilla jakokeskukseen ja suorittamalla tietokoneella tähän tarkoitettuja ohjelmia. Jakokeskus on myös yhdistetty CAN-väylään.

3.2 Ohjausyksiköt

Ohjausyksiköt eli “noodit” ovat kotiautomaatiojärjestelmän ytimiä, jotka sisältävät käytännössä kaikkien laitteiden ohjauslogiikan. Tämä logiikka tallennetaan noodiin tilakoneena, joka ottaa syötteenään CAN-viestejä tai muita ohjaussignaaleja kuten



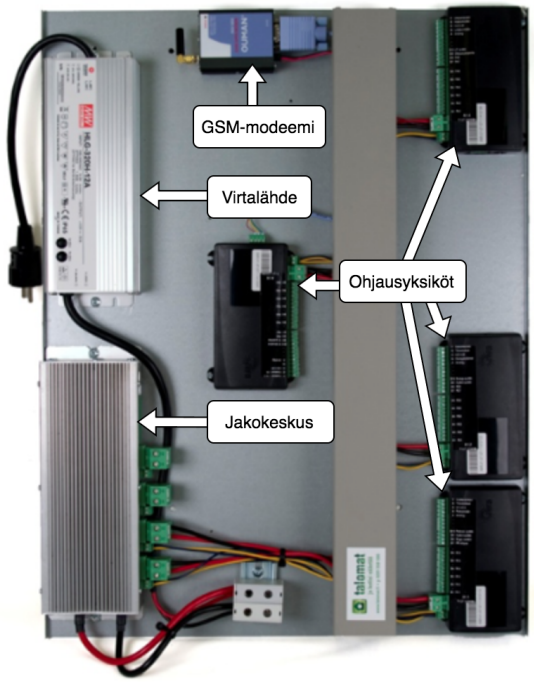
Kuva 3.1 *Kuva taloautomaatiojärjestelmän arkkitehtuurista.*

kytkimien tuloja, sekä ohjaa laitteita lähettämällä edelleen CAN-viestejä tai ohjaamalla suoraan omia lähtöjään. Yhdessä noodissa on 8 tuloa ja 6 lähtöä, sekä lisäksi liitännät CAN-väylään ja RS-232 sarjaväylään GSM-modeemia varten.

Fyysisillä liitännöillä ohjausyksiköt ohjaavat suoraan muun muassa valaistusta. Yhteen ohjausyksikön lähtöön kytkettyyn valaisinryhmään voidaan kytkeä useita valaisimia, mutta tällöin kaikkia valaisimia ohjataan samoilla arvoilla.

Ohjausyksikköön kytketty GSM-modeemi on tarkoitettu hälytystekstiviestien lähettämistä varten. Järjestelmään voidaan määritellä neljä puhelinnumeroa, joihin järjestelmä lähettää viestin esimerkiksi kun murtohälytys laukeaa tai sisälämpötila alittaa hälytysrajan.

Ohjausyksiköt eivät sisällä omaa käyttöliittymäänsä. Päivityksiä ohjausyksiköille tehdään huoltohenkilökunnan toimesta, ja tämä onnistuu lähettämällä jakokeskuksen kautta kaikille järjestelmän ohjausyksiköille päivityksen sarjaliikenteen avulla.



Kuva 3.2 *Kuva järjestelmän sähkökaapista.*

3.3 Kytkimet ja anturit

Järjestelmään kuuluu yksinkertaisena ohjauskäyttöliittymänä seinäkytkimiä. Kytkimet ovat yksinkertaisia palautuvia painokytkimiä, ja niillä voidaan ohjata esimerkiksi valaisinryhmää, valaistustilaa tai liiketunnistinta. Kytkimistä ei ole tällä hetkellä analogista versiota, joten kirkkaudensäätö on toteutettu ohjausyksikön logiikassa. Kun valaisinryhmän kytkintä pidetään pohjassa, valaisinryhmä alkaa ensin muuttua kirkkaammaksi ja takaisin himmeämmäksi saavutettuaan isoimman kirkkauden. Kytkimestä irti päästämällä kirkkaus lukitaan ja tallennetaan muistiin valaisinryhmän seuraavaa päälle kytkemistä varten.

Kytkinten lisäksi järjestelmään kuuluu useita kosteus- sekä lämpötila-antureita. Ohjausyksikkö lukee anturien ilmoittamia lukemia jatkuvasti sekä ilmoittaa niiden muuttumista muuhun järjestelmään CAN-viestien avulla.

Ohjauspääte D92 Ohjauspääte P93

Kuva 3.3 *Kuva vanhoista ohjauspäätteistä.*

3.4 Vanhat ohjauspäätteet

Järjestelmään kuuluu lisäksi yleensä talon eteiseen asennettava ohjauspääte. Ohjauspääte asennetaan seinään ja sillä voidaan ohjata valaistusta, murtohälytintä sekä muita järjestelmään kuuluvia laitteita. Ohjauspäätteeseen kuuluu näyttö, jolla esitetään järjestelmän tilaa sekä saatavilla olevia toimintoja. Järjestelmässä on ollut kahdenlaisia eri ohjauspäätteitä, jotka ovat esitetty kuvassa 3.3.

D92-ohjauspääte on päätteistä vanhempi. Sen ohjauksessa käytetään painokytkimiä, joilla voidaan ohjata valaistustiloja tai siirtyä eri ohjausnäkymiin, kuten murtohälyttimen, auton lämmityksen ja sähkölukkojen ohjaukseen tai muihin asetuksiin. Ohjausnäkymissä valintoja tehdään nuolinäppäimillä, joilla suoritetaan toimintoja, jotka näkyvät näytöllä nuolen vieressä. Osa painikkeista toimii myös numeronäppäiminä, kun murtohälytin- tai puhelinnumeroiden syöttönäkymä on aktiivinen. D92-päätteen näyttö on kaksivärinen LCD-näyttö.

P93-ohjauspääte on päätteistä uudempi. Suurin ero päätteessä vanhaan verrattuna on siinä, että se sisältää kosketusnäytön. Kooltaan se on edeltäjäänsä pienempi, mutta ohjausta varten ei tarvita erillisiä painikkeita. Kosketusnäyttö on resistiivinen ja värillinen LCD-näyttö. P93-pääte sisältää samat toiminnallisuudet kuin D92-pääte, mutta näytön tilan puutteen vuoksi toimintoja on usein sivutettu. Sivujen välillä voidaan navigoida näyttöön ilmestyvien nuolipainikkeiden avulla. Pääte on käytettävyydeltään edeltäjäänsä parempi, koska staattisten ohjauspainikkeiden sijaan painikkeet näytetään ruudulla näkymästä riippuen.

Kuva 3.4 *Kuva uudesta Chipsee-ohjauspäätteestä edestä ja takaa.*

3.5 Uusi ohjauspääte

Asiakas on tilannut järjestelmän laajennusta varten Chipsee-laitteita. Laitteessa on 7 tuuman kapasitiivinen LCD-kosketusnäyttö. Kosketusnäytön kapasitiivisuus mahdollistaa sen, että näyttöä voidaan ohjata sormenpäillä nykyaikaisten älylaitteiden tapaan. Laitteen takapaneelissa on liitännät muun muassa CAN-väylälle ja verkkokaapelille. Laitteeseen on sisäiseen eMMC-muistiin esiasennettu sulautetuille laitteille tarkoitettu Linuxin Ångström-jakelu. Lisäksi laitteessa on microSD-korttipaikka, minkä avulla laitteella voi suorittaa Chipseen jakelemaa Debian-levykuvaa. Levykuva sisältää kaikki laitteelle tarpeelliset ajurit. Laitteella on valmius kehittää Qt-ohjelmia, joista on joitakin esimerkkiohjelmia laitteen eMMC-muistissa. Chipsee-laitteen etu- ja takapaneeleista on esitetty kuvassa 3.4.

4. JÄRJESTELMÄN LAAJENNUS

Tässä luvussa esitellään, mitkä ovat järjestelmän laajennuksen tavoitteet ja kriteerit. Esiteltyjen tavoitteiden ja kriteerien sekä uuden ohjauspäätteen valmiuksien perusteella verrataan kahta vaihtoehtoa, millä teknologioilla laajennus voitaisiin toteuttaa. Lopuksi tehdään vertailun perusteella valinta toteutusteknologiasta.

4.1 Tavoitteet ja kriteerit

Laajennuksen tavoite on korvata taloautomaatiojärjestelmän vanhat ohjauspäätteet uudella. Kun pääte kytketään kiinni järjestelmän CAN-väylään, se osaa tunnistaa järjestelmään kuuluvat laitteet käynnistyksen yhteydessä. Päätteitä on mahdollista yhdistää samaan CAN-väylään useampia ja niitä tulee voida käyttää samaan aikaan, mutta yhteensopivuutta vanhojen päätteiden kanssa ei tarvitse olla. Uuden päätteen tulee sisältää sama toiminnallisuus kuin vanhoissa näytöissä, sekä lisäksi erikseen asiakaspalavereissa päätettyjä toimintoja ja ominaisuuksia.

Tärkein kriteeri uudelle ohjauspäätteelle on olla **nopeampi** ja **helppokäyttöisempi** vanhoihin käyttöliittymiin verrattuna. Tämä on hyvin tärkeää siksi, että kodin toimintojen ohjaamisen pitäisi onnistua ilman järjestelmän käyttöön keskittymistä. Tällöin taataan käyttäjälle paras mahdollinen käyttäjäkokemus, eikä käyttäjä kokisi ohjausjärjestelmää esteenä tai hidasteena kotinsa ohjaamiseen. Tämän saavuttamiseksi sovelletaan moderneja mobiilikäyttöliittymien suunnitteluperiaatteita, kuten yksinkertaisuuden korostamista ja vaadittavien painallusten määrän minimointia. Myös teknologiavalinnalla voidaan vaikuttaa käyttöliittymän tehokkuuteen ja responsiivisuuteen.

Toinen kriteeri ohjauspäätteelle on **reaaliaikaisuus**. Käyttöliittymän tulee päivittyä reaaliajassa, kun järjestelmän antureissa havaitaan muutoksia. Kaikkien anturien, kuten lämpötila-anturien tai valokytkinten tilan muuttumisen täysi reaaliaikaisuus ei ole kriittistä, mutta esim. palohälytystä tai murtohälytyksen päälle kytkeytymistä

ohjauspäätteen tulee ilmaista reaaliajassa. Riittävä reaaliaikaisuus tarkoittaa käytännössä sitä, että käyttäjä ei havaitse järjestelmässä viivettä.

Kolmas kriteeri ohjauspäätteelle on **mahdollisuus ulkoasun kustomointiin**. Asiakkaalla on käytössä graafinen ohjeistus, jonka mukaisiksi ohjelman värit ja tyylit tulisi saada. Kustomoitavuus antaa myös kehittäjälle vapaammat kädet saada ohjelman graafista ilmettä miellyttäväksi, ja tämä on tärkeää hyvän käyttäjäkokemuksen aikaansaamiseksi.

Neljäs kriteeri ohjauspäätteen ohjelmistolle on **modulaarisuus**. Tämä ei näy suoraan käyttäjille, mutta se helpottaa huomattavasti ohjelmiston kehitystä, ylläpitoa sekä laajentamista. Modulaarisuuden avulla voidaan myös tehdä joistain toiminnallisuuksista helposti uudelleenkäytettäviä eri näkymissä. Käyttöliittymässä on tarve esimerkiksi tilaa indikoiville painikkeille, ajastinnäkymille sekä erilaisille viestiikkunoille. Modulaarisuus helpottaa myös käyttöliittymän laajentamista jatkokehityksessä.

Jatkokehitystä ajatellen viides kriteeri laajennukselle on se, että se voidaan laajentaa **yhteensopivaksi pilvipalvelun kanssa**. Projektin ensimmäisessä vaiheessa pilvipalvelua ei tulla toteuttamaan, mutta asiakas on ilmoittanut intresseistä viedä tietyt talon hallinnan ominaisuudet asiakkaan saataville pilvipalvelun kautta. Kehityksessä pitää ottaa huomioon, ettei pilvipalvelun käyttöön siirtyminen vaatisi työläiden integraatiokomponenttien kehittämistä. Tämä on otettava huomioon myös teknologian valinnassa.

4.2 Vaihtoehto 1: alustariippumattomat sovelluskehykset

Ensimmäinen vaihtoehto on toteuttaa laajennus käyttämällä alustariippumattomia sovelluskehyksiä. Käytännössä tämä tarkoittaa sitä, että ohjelmisto kirjoitetaan sovelluskehyksellä, joka käännetään alustalla ajettavaksi binääriohjelmaksi, joka sisältää kaiken sovelluslogiikan käyttöliittymää myöden. Koska ohjelma on käännetty suoraan konekielelle, sitä ajetaan virtuaalikoneen tai moottorin sijaan suoraan prosessorilla.

Yleiset ohjelmointikielet suoraan alustalle käännettäville ohjelmille ovat C ja C++, jotka saadaan käännettyä tehokkaasti konekielelle. Grafiikkakerrosta olisi mahdollista ohjata myös matalalla tasolla esimerkiksi OpenGL:n avulla, mutta toiminnallisen

käyttöliittymän toteuttaminen tällä tavalla on tarpeettoman monimutkaista, työlästä ja aikaavievää. Tätä varten onkin onneksi olemassa sovelluskehyksiä kuten Qt, joilla graafisen käyttöliittymän toteuttaminen on trivialisoitu. Uudella ohjauspäätteellä on myös valmiina valmius ajaa Qt-ohjelmia.

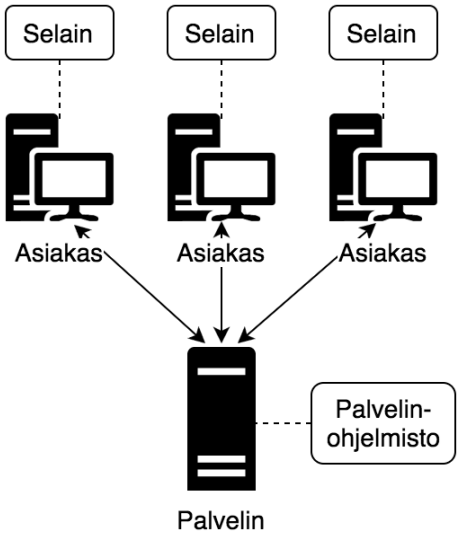
Qt on tehokas C++ -sovelluskehys, jonka avulla voidaan tehdä joustavasti erilaisia käyttöliittymiä. Qt perustuu siihen, että käyttöliittymän layout kuvataan QML-kuvauskielellä. Komponentteihin voidaan kytkeä erilaisia *signaaleja*, jotka välitetään niihin yhdistetyille *slot*-metodeille kun esimerkiksi komponenttia klikataan [20]. Qt tarjosi oman kevyen ikkunointijärjestelmän, QWS:n, aina versioon 4.8 asti, mutta versiossa 5.0 se poistettiin. Tämä pakottaa käyttämään joko jotain kolmannen osapuolen ikkunointijärjestelmää, tai pysymään korkeintaan versiossa 4.8, joka on julkaistu joulukuussa 2011. Versiolle on olemassa myös päivitys 4.8.7 toukokuulta 2015, mutta se ei tarjoa kuin lähinnä virhekorjauksia. QWS on kuitenkin sen verran tärkeä ominaisuus ja X11-palvelinta tehokkaampi, että tuossa versiossa kannattaa pysyä [18].

Qt:lla on mahdollista räätälöidä käyttöliittymän tyylejä CSS:n kaltaisten *style sheettien* avulla. Oletusarvoisesti komponenttien ulkoasua pyritään matkimaan alustan käyttöjärjestelmän tyyleistä, mutta alustariippumattoman käyttöliittymän suunnittelu vaatii kaikkien käytettyjen komponenttien tyylien määrittelyn [19].

4.3 Vaihtoehto 2: web-teknologiat

Web-teknologioilla voidaan toteuttaa *asiakas-palvelin-mallia* noudattavia ohjelmistoja. Keskeisintä asiakas-palvelin-mallissa on sovelluslogiikan toteuttaminen palvelinohjelmistona, ja käyttöliittymän toiminnallisuuden ja esitystavan toteuttaminen asiakasohjelmistona selaimessa [3]. Asiakas- ja palvelinohjelmistojen välinen kommunikointi toteutettaisiin asiakasohjelmiston suorittamilla HTTP-pyynnöillä palvelimeen toteutettavaan REST-rajapintaan tai *full duplex* -kommunikoinnin mahdollistavan WebSocket-yhteyden avulla [11]. Asiakas-palvelin-malli on esitetty kuvassa 4.1.

Web-teknologiapinoon kuuluvat palvelimen ohjelmisto sekä tietokanta, sekä mahdolliset palvelimen laiterajapinnat. Palvelin tarjoaa REST- tai WebSocket-rajapinnan, johon yksi tai useampi asiakas voi ottaa yhteyden selaimen avulla. Selain esittää käyttöliittymän HTML- ja CSS-määrittelyjen avulla, sekä kommunikoi dynaami-

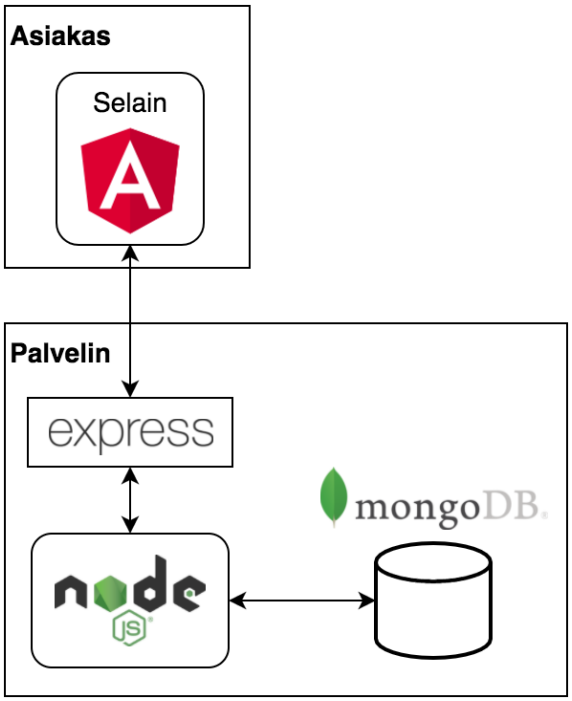


Kuva 4.1 *Asiakas-palvelin-malli.*

sesti palvelimen kanssa JavaScript-ohjelman avulla [9]. Koska sovelluslogiikka on yleensä toteutettu palvelimen puolella, pelkän selainohjelmiston ajaminen ei vaadi paljoa resursseja. Tällä tavalla toteutettua järjestelmää on helppo laajentaa useammalle käyttäjälle. Tämän modulaarisuuden ansiosta on myös suhteellisen vaivatonta luoda uusia käyttöliittymiä esimerkiksi mobiililaitteita varten. Helpoimmillaan uusi käyttöliittymä voidaan toteuttaa pelkästään uusilla HTML- ja CSS-määrittelyillä.

Sekä palvelin- että asiakasohjelmistoja ajetaan usein tulkattavilla kielillä virtuaalikoneessa tai moottorilla. Yleisimpiin palvelinohjelmistojen teknologioihin kuuluvatkin Java, jota ajetaan Javan virtuaalikoneella [1], sekä JavaScript, jota esimerkiksi Node.js:n tapauksessa ajetaan Googlen V8-JavaScript-moottorilla [22]. Ohjelmien tulkattavuuden ansiosta niitä voidaan ajaa myös helposti eri alustoilla, jos alustalle on olemassa virtuaalikoneen tai moottorin toteutus.

Tässä järjestelmässä palvelinohjelmisto kommunikoisi laitteiden kanssa CAN-väylän välityksellä sekä ylläpitäisi järjestelmän tilaa. Palvelin tallentaisi kaikki persistoitavat tiedot, kuten ajastukset, laitteiden nimet sekä käyttäjätilit tietokantaan. Selainohjelmisto ottaisi yhteyttä palvelimeen sen tarjoaman rajapinnan kautta, sekä esittäisi nykyistä järjestelmän tilaa graafisessa käyttöliittymässä mahdollisimman reaaliaikaisesti. Selainohjelmisto lähettää toimintojen suorittamisesta komentoja palvelimelle, joka joko välittää komennot edelleen CAN-väylää pitkin tai suorittaa tietokantakyselyitä.



Kuva 4.2 *MEAN-pino.*

Web-teknologiapino voidaan valita lukuisista kombinaatioista, jotka tyypillisesti sisältävät frontend-kerroksen, rajapintakerroksen, backend-kerroksen sekä tietokantakerroksen. Eräs nykyään suosittu esimerkki on MEAN-pino, joka sisältää MongoDB-dokumenttitietokannan, Express-webrajapintasovelluskehyksen, AngularJS-frontendin sekä Node.js-backendin [14]. Java-pohjaisten backend-sovelluskehysten käyttö on myös yleistä varsinkin yrityskäytössä olevilla palvelimilla.

MEAN-pino on eräs yleinen web-teknologiapino, jonka kaikki kerrokset ohjelmoidaan samalla kielellä: JavaScriptillä. MEAN-pinon käyttöönotto on myös nopeaa ja yksinkertaista, eikä se vaadi paljoa ns. boilerplate-koodia. JavaScript on myös tulkattava kieli, eli sitä voidaan suorittaa JavaScript-moottorilla kääntämättä. Node.js ja MongoDB käyttävät avoimen lähdekoodin V8-JavaScript-moottoria, ja selainohjelmistoa suoritetaan selaimen omalla moottorilla. MEAN-pino on esitetty kuvassa 4.2 [14]

MEAN-pinolla toteutetun projektin hallinta perustuu vahvasti Node.js:n *npm* (Node Package Manager) -paketinhallintaan [22]. Sen avulla on mahdollista ladata yli 300 000 kirjaston joukosta projektin tarpeeseeen sopivia kirjastoja [5]. Esimerkiksi Express on npm-kirjasto, jonka asentamalla voidaan toteuttaa palvelinohjelmistoon

web-rajapintoja HTTP-pyynnöille tai WebSocketeille. MongoDB-tietokannan käsittelyä varten on myös olemassa oma kirjastonsa npm:ssä.

Backend-kirjastojen lisäksi npm tarjoaa myös frontendille työkaluja. Bower on npm:llä asennettava selainkirjastojen paketinhallinta, jolla voidaan lisätä projektiin esimerkiksi AngularJS ja jQuery. Jotta frontend-koodi pysyisi mahdollisimman hyvin hallittavana, npm tarjoaa frontendiä varten käännöstyökaluja, kuten Gulpin. Gulpin avulla voidaan esimerkiksi modularisoida ohjelma erottelemalla sen osat johdonmukaisesti eri tiedostoihin, mutta Gulpin työkaluilla voidaan yhdistää nämä tiedostot yhdeksi tiedostoksi, joka voidaan sisällyttää HTML-tiedostoon. Tällöin ei tarvitse huolehtia jokaisen tiedoston lisäämisestä riippuvuuksien vaatimalla järjestyksellä. Gulpin käännösliukuhihnaan voidaan myös lisätä minifiointi- ja LESS-käännösvaiheet [15].

Taloautomaatiojärjestelmää varten alustan pitää käynnistyessään käynnistää palvelinohjelmisto sekä selain, joka esittää itse käyttöliittymän. Käyttöliittymän esittämiseen laitteen ruudulla on eri vaihtoehtoja. Ilmeisin vaihtoehto on avata web-selain käyttöjärjestelmän käynnistyttyä, mutta graafisen käyttöjärjestelmän käynnistäminen ikkunointijärjestelmineen on raskasta, hidasta ja tässä kontekstissa hyödytöntä. Täyden käyttöjärjestelmän lataaminen voi myös aiheuttaa tietoturvariskejä, esimerkiksi automaattisesti käynnistyvien palveluprosessien ja avautuvien porttien muodossa. Selainta tulisi myös saada muokattua siten, että käyttäjän käytettävissä on vain itse sivun sisältö, eikä esimerkiksi selaimen osoitetta saisi päästä vaihtamaan. Selaimen tehtävän voi hoitaa myös esimerkiksi *Qt Webkit*, jonka saa käynnistettyä omalla ikkunointijärjestelmällään, jolloin raskaampaa käyttöjärjestelmän ikkunointijärjestelmää ei tarvittaisi lainkaan. Ohjauspäätteen valmius Qt-ohjelmiin myös puoltaa päätöstä suorittaa käyttöliittymää yksinkertaisella Qt Webkitiä käyttävällä ohjelmalla [21].

4.4 Vertailu ja valinta

Tässä vertailussa web-teknologioita edustaa teknologiapino, joka perustuu löyhästi MEAN-pinoon. Pinosta kuitenkin päädyttiin korvaamaan MongoDB-tietokanta *sqLite*-tietokannalla, koska jälkimmäinen on relaatiotietokanta, josta kehittäjillä on enemmän kokemusta. Lisäksi Expressin suoraan tarjoaman REST-rajapinnan sijaan käytettäisiin WebSocket-rajapintaa käyttöliittymän reaaliaikaisen päivittämisen mahdollistamiseksi. WebSocket-rajapinta voidaan kuitenkin toteuttaa Expressin

päälle esimerkiksi socket.io -kirjastolla. Toinen vertailukohde on alustariippumattomia sovelluskehyksiä edustava Qt.

Nopeuden kannalta verrattavista vaihtoehdoista etu on Qt:lla. Qt:lla toteutettu ohjelma suoritetaan käännettynä suoraan prosessorilla, joten erikseen käynnistettävillä palvelinohjelmistoilla sekä selaimilla JavaScript-moottoreineen ovat nopeudessa eri luokassa. Käytettävä alusta on kuitenkin riittävän tehokas molemmilla vaihtoehdoilla toteutettujen ohjelmistojen ajamiseksi, ja mahdolliset nopeuserot toteutustapojen välillä tulevat olemaan lähinnä marginaalisia, eivätkä ne todennäköisesti näy käyttäjälle ollenkaan. Web-vaihtoehdon nopeus riippuu kuitenkin myös siitä, kuinka tehokas JavaScript-moottori on käytettävissä Qt Webkitillä.

Helppokäyttöisyys voidaan saavuttaa hyvällä käyttöliittymäsuunnittelulla, mikä ei suoraan liity teknologiavalintoihin. Suunnittelun lisäksi helppokäyttöisyyteen ja yleiseen käytettävyyteen vaikuttavat käyttöliittymän vasteajat. Tässä mielessä Qt on tehokkaampana vaihtoehtona parempi vaihtoehto. Edelleen kuitenkin erot ovat teknologioiden välillä sen verran pieniä, että käytettävyyskriteeri täyttyy varmasti molempien teknologioiden osalta, varsinkin kun verrataan edelliseen järjestelmään.

Käyttäjän havaitseman reaaliaikaisuuden aikaansaaminen on triviaalia Qt:lla, koska ohjelmistossa välitettäviä signaaleja voidaan kytkeä minkä tahansa komponenttien slot-metodeihin, joilla voidaan päivittää komponentin arvot pienillä viiveillä. Tämä tarkoittaa kuitenkin sitä, että jokainen reaaliaikaisesti päivittyvä komponentti pitää liittää erikseen vastaaviin signaaleihin, mikä saattaa heikentää koodin selkeyttä. Web-vaihtoehdolla voidaan välittää järjestelmän tilan muutoksesta tapahtuma käyttöliittymälle WebSocketin avulla, jolla saadaan myös reaaliaikaisuus saavutettua. Vaihtoehdoissa ei ole siis käytännössä juuri eroa reaaliaikaisuuden kannalta, eli molemmilla vaihtoehdoilla kriteeri saadaan täytettyä.

Molemmilla vaihtoehdoilla on mahdollista kustomoida käyttöliittymän ulkoasua. Web-vaihtoehdoilla tämä voidaan tehdä CSS:n avulla. CSS mahdollistaa komponenttien ulkoasun ja sijoittelun määrittelemisen lisäksi niiden animoinnin. CSS:n avulla mille tahansa komponentille voidaan luoda alusta asti oma tyylinsä, mikä saadaan sopimaan asiakkaan graafiseen ohjeistukseen. CSS toimii inspiraation lähteenä Qt:n style sheeteille. Syntaktisesti nämä ovat hyvin lähellä toisiaan, mutta Qt:lla on mahdollisuus muokata vain osaa ulkoasuominaisuuksista CSS:ään verrattuna. Web-vaihtoehdolla on käytössä myös osa CSS3:n ominaisuuksista, kuten varjostukset sekä web-fontit. Nämä ovat saatavilla myös Qt:lla, mutta ne pitää toteuttaa ohjelmakoo-

dissa, mikä vaikeuttaa tyylien muokkaamista ja hallintaa. Qt:lla komponentit matkivat myös oletusarvoisesti käyttöjärjestelmän ulkoasua, mikä ei ole tavoitteena. CSS on jonkin verran siis Qt style sheetejä joustavampi vaihtoehto, joten web-vaihtoehto on tässä mielessä parempi vaihtoehto kriteerin täyttämiseksi.

Modulaarisuus saadaan ohjelmakoodin tasolla toteutettua molemmilla vaihtoehdoilla. AngularJS:llä voidaan toteuttaa komponentteja uudelleenkäytettävinä *direktiiveinä* [4], kun taas Qt:lla tämä tehtäisiin luokkien avulla. Käännettävyytensä takia Qt on kuitenkin vaihtoehdoista joustamattomampi. Esimerkiksi ohjelmapäivityksen lataaminen laitteelle vaatisi ohjelman uudelleenkääntämisen, jollei laitteet satu olemaan kokoonpanoltaan täsmälleen samanlaisia, jolloin vain ajettavan ohjelman korvaaminen uudella binääritiedostolla riittäisi. Käytännössä tämä kuitenkin todennäköisesti aiheuttaa ongelmia, koska ei ole takeita siitä, että laitteiden kokoonpano ei tulisi muuttumaan. Web-vaihtoehdon tapauksessa päivitys voidaan tehdä JavaScript-koodien korvaamisella ja ohjelmien uudelleenkäynnistämisellä, joten se hoituisi huomattavasti helpommin. Qt-vaihtoehto vaatisi käytännössä oman päivitysasentajan, joka kääntäisi ohjelman alustalle. Web-vaihtoehto suoriutuu siis tästä kriteeristä parhaiten.

Web-vaihtoehto on sellaisenaan valmis laajennettavaksi yhteensopivaksi pilvipalvelun kanssa, koska palvelinohjelmisto on jo siihen valmiina. Rajapinta pilvipalvelun kanssa kommunikointia varten pitäisi toteuttaa erikseen, mutta tämä voidaan ajaa samalla palvelinohjelmistolla. Myös käyttöliittymän toteutusta voidaan uudelleenkäyttää pilvipalvelussa. Qt-vaihtoehdon tapauksessa alustalla pitäisi käynnistää erillinen web-palvelinohjelmisto ja rajapinnat palvelimelta pilvipalveluun sekä myös rajapinta palvelimen ja Qt-ohjelman välille. Tällaisen rakenteen Qt-ohjelman päälle rakentaminen on työlästä, eikä käyttöliittymän toteutusta voida pilvipalvelun web-ympäristössä uudelleenkäyttää sellaisenaan, vaan web-ohjelma pitäisi toteuttaa uudelleen. Tämä on omiaan vaikeuttamaan myös ohjelman ylläpidettävyyttä, kun käytännössä samalle käyttöliittymälle on olemassa toteutukset eri teknologioille. Web-vaihtoehto täyttää tämän kriteerin selvästi paremmin.

Edellä käytyjen kriteerien lisäksi on otettava myös huomioon kehittäjien kokemus, jota on jo ennestään enemmän web-teknologioista. Erillisen palvelin- ja selainohjelmien ajaminen samalla alustalla voi kuulostaa resursseja vievältä, mutta se palvelee enemmän jatkokehitystä verrattuna pelkkään Qt-ohjelmaan. Asiakas on myös ilmaissut kiinnostuksen kohteekseen pilvipalvelun kehittämisen tulevaisuudessa, mikä

puoltaa myös web-vaihtoehdon valintaa. Tämän vertailun tuloksena päädyttiin toteuttamaan järjestelmän laajennus web-vaihtoehdolla. Web-käyttöliittymän esittämistä varten toteutetaan kuitenkin Qt:lla yksinkertainen selainohjelma Qt Webkitin avulla.

5. LAAJENNUKSEN TOTEUTUS

Tässä luvussa esitellään laajennukseen toteutettavat toiminnallisuudet, minkä jälkeen käydään läpi järjestelmän suunnittelua arkkitehtuurien avulla. Luvussa esitellään myös kehitykseen oleellisia huomioita sekä työkaluja. Lopuksi backend- ja frontend-ohjelmien ratkaisut ja tarkemmat arkkitehtuurit käydään läpi omissa aliluvuissaan.

5.1 Toteutettavat toiminnallisuudet

Laajennukseen toteutetaan seuraavat toiminnallisuudet. Osa toiminnallisuuksista on samoja, mitkä löytyvät jo vanhasta järjestelmästä, ja osa toteutetaan uusina. Osaa toiminnallisuuksista rajaavat järjestelmässä käytössä olevat CAN-komennot, joten esimerkiksi valaistustilat ovat rajattu kolmeen eri vaihtoehtoon.

Valaistustilojen valinta. Käyttäjä voi kytkeä käyttöliittymästä päälle ohjausyksiköihin valmiiksi ohjelmoituja valaistustiloja, jotka ohjaavat useiden sisävalaisimien tiloja ja himmennysarvoja. Valaistustiloja ovat *perus*, *tunnelma* ja *yö*.

Ulkovalojen ohjaaminen liiketunnistuksella ja hämäräkytkimillä. Käyttäjä voi valita, käytetäänkö ulkovalaisimien päällekytkemiseen liiketunnistimia tai hämäräkytkimiä.

Viikkoajastusten valinta. Käyttäjä voi valita ulkovaloille, pistorasioille sekä lämmittimille ajastuksia, mitkä riippuvat viikonpäivästä ja kellonajasta.

Useiden kytkinten käsinohjaus. Käyttäjä voi kytkeä päälle ja pois erilaisia kytkimiä, kuten yksittäisiä valaistusryhmiä, pistorasioita, lämmittimiä, autonlämmittimiä ja ilmastoinnin tehostamista.

Hälytysjärjestelmän ohjaus. Käyttäjä voi kytkeä hälytysjärjestelmän päälle syöttämällä koodin poistuessaan talosta. Koodi tarkistetaan, ja jos se on oikein,

kytketään hälytysjärjestelmä päälle asetetulla poistumisviiveellä. Hälytys voidaan purkaa syöttämällä koodi uudestaan. Jos liiketunnistimet tunnistavat liikettä hälytyksen ollessa päällä, annetaan käyttäjälle tietty aika syöttää koodi ennen hälytyksen laukaisemista.

Kellon ajan ja anturien arvojen näyttäminen. Laitteen ylläpitämä kellonaika sekä sisä- ja ulkolämpötila-anturien lukemat esitetään näkymän ylälaidassa. Muut järjestelmän valvomien anturien, kuten kosteusanturien ja muiden lämpötila-anturien lukemat löytyvät vastaavilta näkymiltä.

Sähkölukkojen ohjaaminen. Käyttäjä voi avata ja lukita järjestelmään mahdollisesti liitettyjä sähkölukkoja. Lukittujen lukkojen lukumäärä ja nimet esitetään näytöllä.

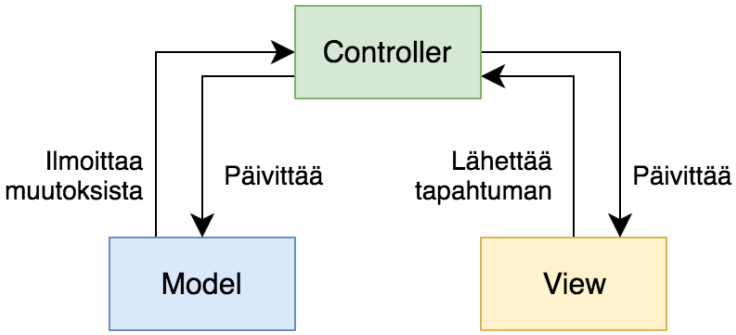
Asetusnäkymä. Käyttäjä voi kirjautua laitteen asetuksiin tunnuksillaan, ja tehdä sen kautta järjestelmän asetuksiin muutoksia. Tällaisia muutoksia ovat esim. laitteiden ja anturien uudelleennimeäminen, murtohälyttimen koodin vaihtaminen, lokitietojen selaaminen sekä puhelinnumeroiden asetus ilmoituksia varten.

5.2 Suunnittelu

Ohjelmisto on suunniteltu käyttämällä MVC-mallia perustana. MVC on kolmijakoinen ohjelmistosuunnittelun malli, jonka avulla saadaan eroteltua käyttöliittymälogiikka varsinaisesta tallennettavasta raakadatasta. MVC on lyhenne sanoista **model** (malli), **view** (näkymä) ja **controller** (ohjain). Malli käsittää ohjelmiston tietovaraston, näkymä esittää datan käyttäjälle esitettävässä muodossa sekä ohjain tekee muunnokset mallin ja näkymän välillä. Ohjain tekee myös muutokset malliin näkymän tapahtumien mukaisesti. MVC-malli on esitettynä kuvassa 5.1. [13]

MVC-mallin avulla voidaan tehdä selkeä erotus sovelluslogiikan datan ja näkymän välille. MVC-malli on myös helppo ottaa käyttöön valittujen teknologioiden kanssa, koska AngularJS sisältää ohjaimen käsitteen, ja näkymäkomponentteina toimivat joko valmiit HTML-elementit tai AngularJS-direktiivit, jotka ovat atomisia HTML-elementtien yhdistelmiä [4]

Järjestelmän tila säilytetään mallissa, mutta koska taloautomaatiojärjestelmän tila muuttuu seinäkytkimien sekä ohjausyksiköiden logiikan mukaan, sitä ei ole järkevää persistoida tietokantaan. Tämän sijaan ohjauspäätteen käynnistyessä suoritetaan



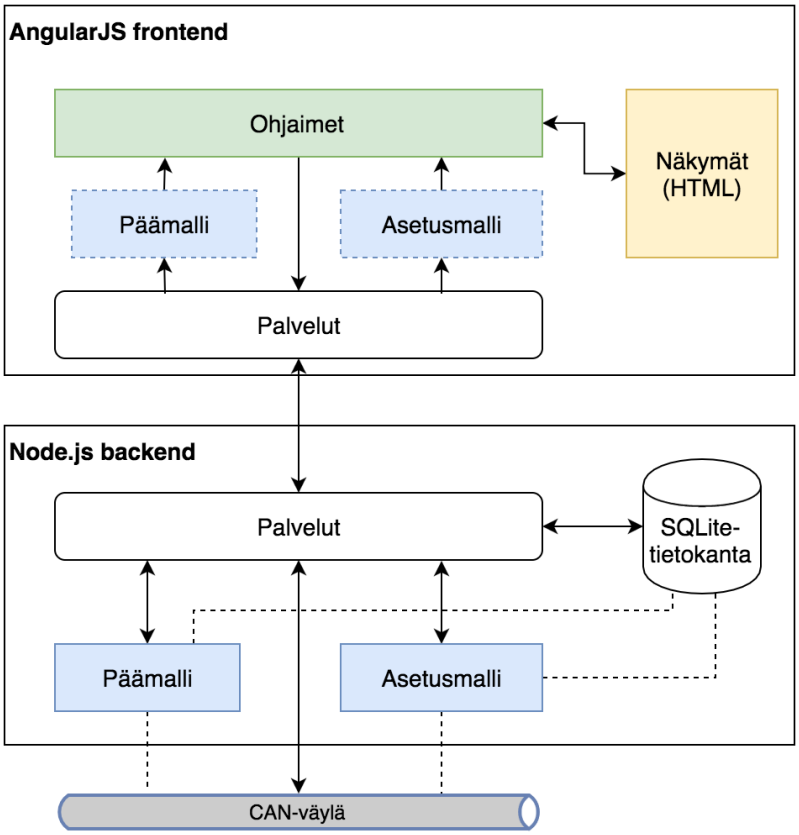
Kuva 5.1 *Kuva MVC-arkkitehtuurista.*

CAN-väylällä kyselyt, joilla selvitetään järjestelmän senhetkinen tila. Tämän jälkeen tilan mallia voidaan päivittää aina muutosten tapahtuessa.

Ohjauspääte sisältää kuitenkin järjestelmän reaaliaikaisen tilan lisäksi myös dataa, joka on ohjauspäätekohtaista, eikä suoraan liity muuhun taloautomaatiojärjestelmään. Tällaista dataa ovat esimerkiksi käyttäjän antamat nimet laitteille sekä ajastussäännöt. Tämä data tulee myös tallentaa malliin, mutta se pitää lisäksi persistoida tietokantaan.

Koska asetusnäkymään vaaditaan autentikointia, sen käyttämä data eritellään omaan malliolioonsa, joka palautetaan backendiltä vain käyttäjän ollessa kirjautuneena sisään. Mallien avulla voidaan myös lähettää tapahtumia frontendin ja backendin välillä, sekä autentikointia vaativat komennot, kuten murtohälyttimen koodin vaihtaminen, käsitellään asetusmallissa. Asetusmalli toimii ns. päämallin kanssa samaan tapaan samojen palveluiden kautta.

Kuvassa 5.2 kuvataan laajennuksen yksinkertaistettua ohjelmistoarkkitehtuuria, josta voidaan nähdä helposti MVC:n vaikutus arkkitehtuuriin. Mallin jakautuminen persistoitavaan ja ajonaikaiseen osuuteen sekä niiden riippuvuudet CAN-väylästä ja tietokannasta ovat esitetty katkoviivoilla. Tosiasiassa malleja ylläpidetään backendissa eri palveluiden (“services”) avulla. Molemmat mallit replikoidaan frontendille käytännön syistä, koska muuten useita yksittäisiä järjestelmän tilatietoja pitäisi kysellä erikseen jatkuvasti backendin puolelta, mikä toisi viivettä ohjelmistoon ja ruuhkauttaisi palvelinta turhaan. Kaksisuuntaisen rajapinnan avulla on yksinkertaista ylläpitää replikoituja malleja ajan tasalla lähettämällä päivityspyyntö backendilta frontendille, kun järjestelmän tila muuttuu. Frontendin ylläpitämät replikoidut mallit ovat kuitenkin vain lukukäytössä, eikä niitä muokkaamalla voida tehdä

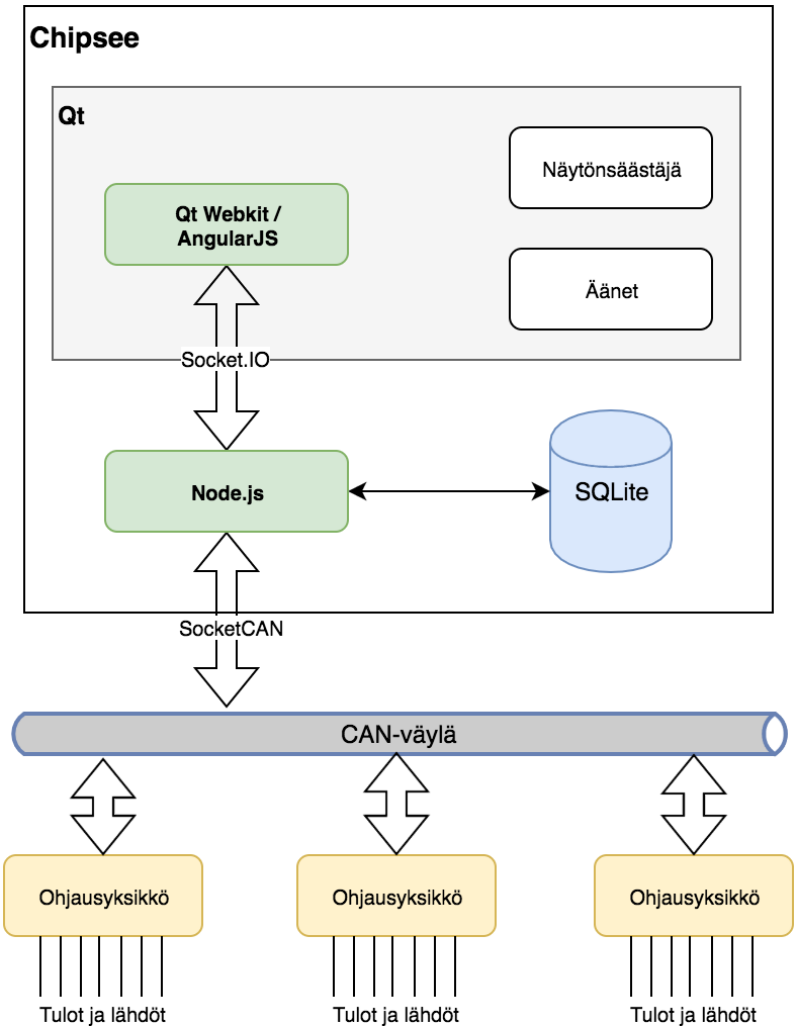


Kuva 5.2 *Yksinkertaistettu kuva laajennuksen ohjelmistoarkkitehtuurista.*

mallin päivityksiä backendille. Mallin päivitys frontendiltä käsin onnistuu kuitenkin lähettämällä ohjaimilta rajapintaa pitkin päivityspyynnön, jolloin backendin mallit päivittyvät ja päivitys propagoituu myös takaisin frontendin replikoiduille malleille.

Selainohjelman esittämistä varten kehitetään lisäksi Qt-ohjelma, joka esittää käyttöliittymän Qt Webkit -selainnäkymän avulla. Qt-ohjelman avulla olisi myös mahdollista hallita ohjauspäätteen näytönsäästäjää sekä ääniä toteuttamalla Qt-ohjelmalle funktioita, joita voidaan kutsua Qt Webkitin JavaScript-moottorin kautta. Laajennuksen järjestelmäarkkitehtuuri rajapintoineen on esitetty kuvassa 5.3.

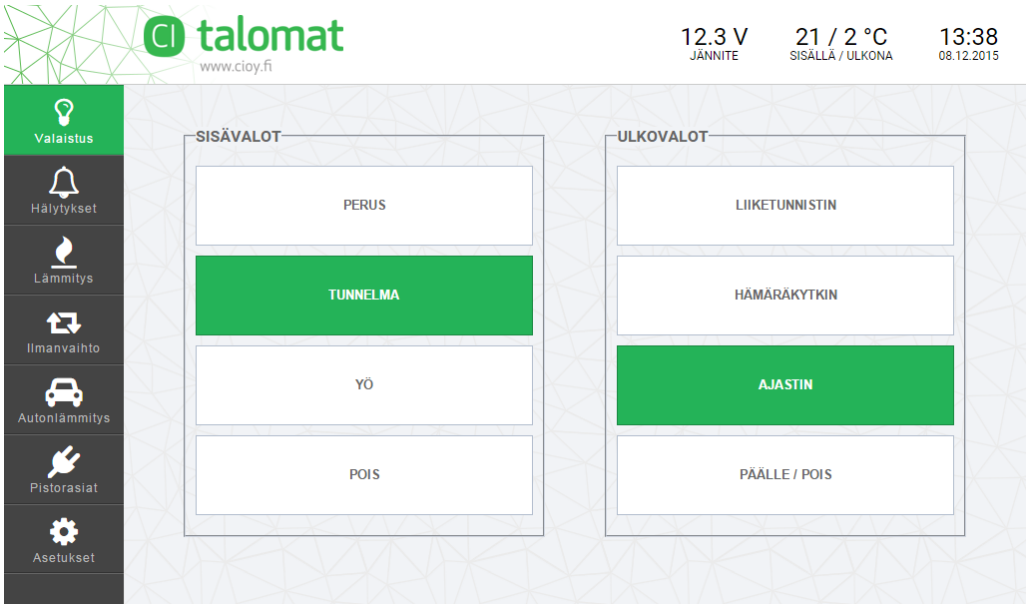
Käyttöliittymäsuunnittelussa pyrittiin toteuttamaan käyttöliittymä, mikä vastaa modernin kosketusnäyttökäyttöliittymän käsityksiä ja vaatimuksia. Ohjauspäätteen kapasitiivisen kosketusnäytön ansiosta tämä on mahdollista, mutta sen takia myös tilankäytön huolellinen suunnittelu korostuu, koska näyttöä ohjataan sormenpäillä kosketusnäyttökynän tai kynnen sijaan. Painikkeista on tehtävä tarpeeksi suuria, jotta kuka tahansa käyttäjä voi ohjata ohjauspäätettä ilman virheosumia.



Kuva 5.3 *Kuva laajennuksen järjestelmäarkkitehtuurista.*

Käyttöliittymässä tulee myös huomioida se, että käytössä ei ole esimerkiksi tekstin kirjoitukseen tai murtohälyttimen koodin syöttämistä varten erillistä fyysistä näppäimistöä. Tämän takia käyttöliittymään toteutetaan virtuaalinäppäimistö, joka tulee esiin kun käyttäjä valitsee tekstikentän. Murtohälyttimen koodin syöttämistä varten toteutetaan virtuaalinäppäimistö pelkillä numeronäppäimillä. Lisäksi on toteutettava tapa, millä käyttäjä voi vierittää ruutua, jos näkymän sisältö ei mahdu kerralla ruutuun. Yksinkertaiset sivunvaihto- tai vierityspainikkeet olisivat helppoja toteuttaa, mutta käytettävyyden takia käyttöliittymään toteutetaan pystysuora vieritys sormella pyyhkäisemällä nykyaikaisten kosketusnäyttölaitteiden tapaan.

Asiakkaalla on käytössään graafinen ohjeistus, mikä vaikuttaa käyttöliittymän värei-



Kuva 5.4 *Näytönkaappaus valaistusnäkymästä.*

hin ja yleiseen ilmeeseen. Graafinen ohjeistus sisältää myös fontteja, mutta käyttöliittymässä päädyttiin käyttämään yleisiä fontteja, mitkä ovat helpommin luettavissa pienemmilläkin fonttiko'oilla. Käyttöliittymän taustakuviot ovat myös peräisin graafisesta ohjeistuksesta. Näytönkaappaus valaistusnäkymästä on esitetty kuvassa 5.4.

5.3 Kehitys

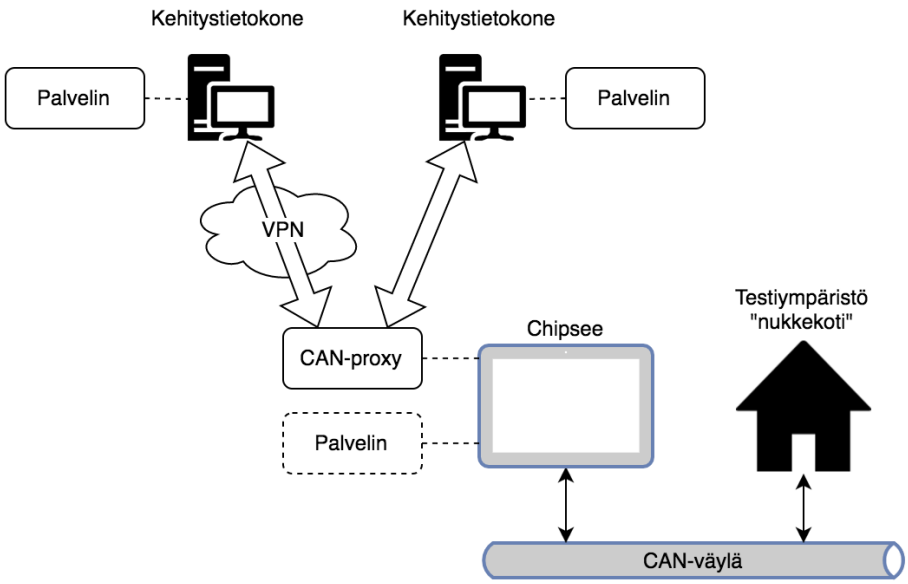
Ohjelmistokehitys tapahtuu pääsääntöisesti omalla tietokoneella. Backend- ja frontend-ohjelmistot ajetaan omina ohjelminaan: backend käynnistetään suoraan JavaScript-lähdekoodista Node.js:llä, mutta frontend-ohjelmisto ajetaan Gulp-liukuhihnan avulla. Gulpin avulla frontend voidaan Node.js-ohjelman tavoin modularisoida useisiin eri lähdekooditiedostoihin, jotka Gulp yhdistää muunnosten avulla yhdeksi JavaScript-tiedostoksi selainta varten. Tämä vähentää huomattavasti tarvetta tehdä ylimääräisiä HTTP-pyyntöjä skriptitiedostojen hakemiseksi. Gulp huolehtii myös skriptitiedostojen välisestä riippuvuudesta – ilman Gulpia tästä pitäisi huolehtia HTML-tiedostossa skriptien latausjärjestystä muuttamalla. Gulp suorittaa myös minifiointia, minkä avulla ohjelmiston koodi saadaan tiivistettyä mahdollisimman paljon. Tämä vähentää ohjelman latausaikaa sekä parantaa myös jonkin verran suorituskykyä selaimen JavaScript-moottorissa.

Asiakas toimitti kehitystä varten Chipsee-alustat, jolle kehitetään lisäksi Qt-sovellus selainnäkymää varten. Alustalle tehdään myös muuta valmistelua ohjelman ja tarpeellisten palveluiden käynnistämistä varten, sekä sillä voidaan kommunikoida CAN-väylän kanssa. Ohjelma voidaan siirtää kehitysvaiheessa alustalle verkkoyhteyttä pitkin tai SD-kortin avulla. Tämä on kuitenkin hyvin hidasta, koska joka muutoksen jälkeen tulisi ohjelma kääntää ja siirtää laitteelle käsin. Tämän takia kehityksen alkuvaiheessa toteutettiin yksinkertainen Qt Webkit -ohjelma, jolle annetaan esitettävän sivuston URL käynnistysparametrina. Tämän ansiosta itse selainohjelmisto voidaan palvella kehitystietokoneelta, ja Qt-ohjelma esittää ohjelmiston Chipsee-alustalla.

Myös palvelinohjelmisto voi olla kehitysvaiheessa ajossa kehitystietokoneella, mutta CAN-yhteyden saa otettua vain Chipseen kautta, koska käytössä ei ole CAN-liitäntää tarjoavaa oheislaitetta kehitystietokoneille. Etätyöskentelyä varten Chipsee-alustalle kehitettiin yksinkertainen CAN-proxyohjelma, jonka avulla saadaan välitettyä lähetettävät ja vastaanotettavat CAN-viestit backendin ja Chipseen välillä. Tämä ohjelma ajetaan Chipseellä ja on toteutettu Node.js:llä. Ohjelma tarjoaa Socket.IO-rajapinnan, jota pitkin CAN-viestit kulkevat backendin ja Chipseen välillä kaksisuuntaisesti. Palvelinohjelmistoa käynnistettäessä voidaan määritellä käynnistysparametreilla käytetäänkö oikeaa CAN-väylää vai proxya, jolloin annetaan käynnistyksessä lisäksi proxyohjelman osoite. CAN-proxyn toimintaa kehitysympäristössä esitetään kuvassa 5.5. Lisäksi Chipseellä ajetaan yksinkertaista CAN-väylämonitoriohjelmaa, jonka avulla voidaan seurata CAN-viestiliikennettä ja varmistaa, että se toimii oikein.

Chipsee-alustan lisäksi asiakas toimitti kaksi erilaista testiympäristöä, jotka ovat alunperin valmistettu lähinnä messuesittelyitä varten. Toinen ympäristö on “nukkekoti” ja toinen seinälle ripustettava taulu. Molemmissa ympäristöissä on useita LED-valaisinryhmiä sekä järjestelmään kuuluvia antureita. Ympäristöihin kuuluvat myös ohjauspäätteen edellisiä versioita sekä järjestelmää ohjaavat ohjausyksiköt, joiden kanssa pääte kommunikoi CAN-väylän välityksellä. Tämä mahdollistaa sen, että ohjauspäätteen ja CAN-viestien toimintaa voidaan testata kehityksenaikaisesti todenmukaisessa ympäristössä.

Kehitys on riippuvainen siitä, että toimiva yhteys jompaankumpaan testiympäristöön on olemassa. Käytännössä kaksi testiympäristöä mahdollistaa korkeintaan kahden kehittäjän samanaikaisen työskentelyn. Testiympäristöiden välillä on lisäksi pieniä ominaisuuseroja, joten välillä ympäristöjä pitää vaihtaa, jotta toiminnallisuus



Kuva 5.5 *CAN-proxyn toiminta kehitysympäristössä.*

voidaan varmentaa. Tämän takia oli ajatuksena kehittää simulaattori, jolla voitaisiin luoda minkälainen virtuaalitestiympäristö tahansa, ja joka käsittelisi myös CAN-viestit samaan tapaan kuin oikeat ympäristöt. Simulaattorin kehittäminen kuitenkin olisi vaatinut sen verran aikaa, että sen kehitys ei mahtunut projektin aikatauluun.

5.4 Käyttöliittymän kuvaus

Kun käyttäjä alkaa käyttää ohjauspäätettä, käyttöliittymä esittää oletusarvoisesti päänäkymän. Päänäkymä koostuu *etu-*, *valaistus-*, *hälytys-*, *lämmitys-*, *ilmanvaihto-*, *autonlämmitys-* sekä *pistorasiasivuista*. Käyttäjät tai huoltohenkilökunta voivat kirjautua tunnuksillaan asetusnäkymään, joka sisältää omat näkymäsivunsa: *käyttäjät*, *ilmanvaihto*, *laitteet*, *loki*, *puhelinnumerot* ja *yleiset*.

Etusivu on oletusnäkymä, mihin siirrytään näytönsäästäjästä. Etusivulla käyttäjä voi käyttää yleisimmin käytettyjä toimintoja, eli vaihtaa valaistustilaa tai asettaa murtohälyttimen päälle. Murtohälytin voidaan asettaa täyteen valvontaan, kuorivalvontaan tai pelkän lisärakennuksen (usein autotalli) valvontaan. Toiminnot ovat esitettynä painikkeilla, joilla voidaan ilmaista toiminnon nykyistä tilaa painikkeen taustavärin avulla – vihreä tarkoittaa päällä ja valkoinen tarkoittaa pois päältä. Näitä tilapainikkeita käytetään melkein jokaisessa ohjauspäätteen näkymässä.

Valaistusnäkymä koostuu kahdesta sivusta, joista ensimmäisellä käyttäjä voi ohjata valaistustilojen lisäksi myös ulkovaloja sekä sitä, kytkeytyvätkö ulkovalot päälle liiketunnistimen tai hämäräkytkimen avulla automaattisesti. Ulkovaloille on myös painike, jolla avataan dialogi-ikkuna viikkoajastuksien määrittämistä varten. Viikkoajastuksiin voidaan lisätä ajastussääntöjä, joista kukin sisältää viikonpäiviä sekä alku- ja loppuajan, jolloin ulkovalojen tulisi olla päällä. Lisäämällä useita ajastussääntöjä voidaan esimerkiksi määritellä eri aikavälit arkipäiville ja viikonlopulle. Risteäviä ajastussääntöjä ei kuitenkaan sallita, joten ne tarkistetaan backendin puolella ja ilmoitetaan tarvittaessa virheestä.

Hälytysnäkymässä voidaan kytkeä päälle murtohälytyksiä samaan tapaan kuin etusivulla. Lisäksi hälytysnäkymään on sijoitettu magneettiventtiilin manuaalinen ohjaus. Lisäksi näkymässä voidaan asettaa lämpötilahälytykselle lämpötilaraja. Jos annettu raja alittuu, ohjausyksikkö lähettää GSM-moduulin kautta tekstiviestit asetuksissa määrätyille vastaanottajille.

Valaistusnäkymän toisella sivulla ovat painikkeet, joilla pääsee kunkin valaistustilan uudelleenmäärittelytilaan, sekä kytkin jokaiselle järjestelmään määritetylle valaisinryhmälle. Valaisinryhmille on myös kirkkaudensäätöä varten liukukytkin, mutta ohjausyksiköiden puutteellisen toiminnon takia kirkkaus pitää tallentaa erikseen. Kirkkaus voidaan kyllä asettaa valaisinryhmälle CAN-käskyllä arvoon 0-255, mutta tällöin kirkkauden arvo ei kuitenkaan tallennu muistiin, vaan edellinen kirkkaus otetaan käyttöön, kun valaisinryhmä kytketään pois ja uudestaan päälle. Ohjausyksiköt tallentavat kirkkauden vain, jos komento on sama mitä lähetetään seinäkytkinten käytöstä – kytkintä pidetään pohjassa, kunnes haluttu kirkkaus saavutetaan. Ainoa vaihtoehto tallentaa liukukytkimen arvo ohjausyksikköön oli simuloida tällaista painallusta, eli lähetetään CAN-viesti alaspainetusta kytkimestä, odotetaan tarvittava aika ja lähetetään toinen CAN-viesti kytkimestä irti päästämisestä.

Lämmitys-, autonlämmitys- ja pistorasiasivut ovat käytännössä samankaltaiset näkymät. Sivulla esitetään jokaiselle järjestelmään määritetylle lämmitys-, autonlämmitys- tai muulle pistorasialle manuaalinen kytkinohjaus sekä kaksi tapaa tehdä niille ajastuksia – viikkoajastus tai viiveajastus. Viikkoajastusten määrittely on samanlainen kuin ulkovaloilla. Viiveajastuksella voidaan määritellä kertaluontoisesti aikaväli, minkä ajaksi laite kytketään päälle. Viiveajastukselle annetaan tunti- ja minuuttimäärä, minkä ajan päästä laite halutaan käynnistää, sekä tunti- ja minuuttimäärä, kuinka kauan halutaan, että laite on päällä. Käyttöliittymä esittää

myös valintojen mukaiset käynnistymis- ja sammumiskellonajat, jotka ovat käyttäjälle hyödyllistä tietoa esimerkiksi autonlämmityksessä, eikä niitä tarvitse tällöin laskea erikseen.

Ilmanvaihtonäkymässä esitetään järjestelmän kaikkien lämpötila- ja kosteusanturien lukemat. Lisäksi näkymässä voidaan kytkeä ilmastoinnin tehostus päälle ja pois kytkimen avulla.

Näkymäsivujen lisäksi järjestelmän tilaa ilmaistaan myös yläpaneelin avulla, mikä on kaikille näkymille yhteinen. Yläpaneeli näyttää kellonajan sekä ulko- ja sisälämpötilan. Poikkeustilanteissa kuten palo-, häkä ja kosteushälytyksissä tilaa ilmaistaan ikonilla ja ruutuun avautuvalla hälytysikkunalla. Yläpaneelissa on lisäksi ilmaisin, joka kertoo montako järjestelmään kytkettyä ovea talossa on auki. Auki olevien ovien nimet saadaan näkyville kun ovi-ikonia kosketetaan. Ainoa ohjaus, jota voidaan tehdä yläpaneelin kautta, on sähkölukkojen ohjaus. Lukkoikonin painaminen avaa sähkölukkonäkymän, josta voidaan ohjata joko kaikkia tai yksittäisiä järjestelmään kytkettyjä sähkölukkoja. Lisäksi magneettiventtiilin ollessa suljettuna yläpaneelista voidaan avata venttiili painamalla sitä

Asetusnäkymään kirjaudutaan omasta näkymästään, jossa valitaan käyttäjänimi listasta, minkä jälkeen käyttäjältä pyydetään salasanaa. Käyttäjänimi voidaan kirjoittaa myös käsin, jolloin huoltohenkilökunta pääsee kirjautumaan laitteen asetuksiin erillisillä tunnuksilla. Listavalintaan päädyttiin asiakkaan kanssa, koska käyttäjänimet koostuvat käytännössä talon asukkaista, eikä niitä ole syytä salata. Nimen valitseminen listalta myös nopeuttaa huomattavasti asetusnäkymään kirjautumista. Käyttäjätunnuksen ja salasanan syöttöä varten esitetään virtuaalinäppäimistö, kun käyttäjä valitsee tekstikentän. Virtuaalinäppäimistö esitetään melkein koko ruudun kokoisena, jolloin valittu tekstikenttä ja siihen liittyvä otsikko tai kysymys voivat jäädä piiloon. Tämän takia tekstikenttä sekä sen otsikko esitetään aina myös näppäimistön yläpuolelle, kun näppäimistö on auki. Näppäimistö on muista käyttöliittymäkomponenteista poiketen DOM-puun juuressa sekä kutsutaan globaalista JavaScript-scopesta, koska näppäimistönä käytetään hieman muokattua avoimen lähdekoodin näppäimistöä, sekä sen havaittiin olevan liian työläs käännettäväksi AngularJS-direktiiviksi.

Käyttäjät-asetussivulla voidaan hallita järjestelmän käyttäjiä, ketkä pääsevät kirjautumaan asetusnäkymään sisään. Yksi käyttäjistä on kuitenkin aina pääkäyttäjä, jota ei voida poistaa. Lisäksi huoltotunnuksia ei näytetä sivulla lainkaan. Näkymäs-

sä voidaan käyttäjien poistamisen ja lisäämisen lisäksi vaihtaa salasanoja (omaa tai myös muiden riippuen oikeuksista) sekä nollata pääkäyttäjän salasana.

Ilmanvaihdon asetuksissa voidaan ohjata päänäkymän ilmanvaihtosivun tapaan ilmastoinnin tehostusta käsin, mutta lisäksi tehostukselle voidaan asettaa automaattiset käynnistysehdot. Ehdot voidaan asettaa lämpötila- ja kosteusanturien arvoilla liukukytkimillä. Tehostukselle voidaan asettaa liukukytkimellä myös viive, kuinka monen minuutin päästä tehostus käynnistetään ehtojen toteutuessa. Näkymässä voidaan myös säätää ulko- ja sisälämpötila-antureiden poikkeamia asteen tarkkuudella.

Laitteiden asetussivulla käyttäjä voi antaa nimen jokaiselle valaisinryhmälle, anturille, valaistustilalle, ovelle sekä muulle järjestelmään kytketylle laitteelle. Oletusarvoisesti kaikki laitteet esitetään käyttöliittymässä vain järjestelmän ID-numeroina, mutta uudelleennimeämisen avulla järjestelmää voidaan ohjata huomattavasti helpommin.

Lokinäkymässä voidaan seurata järjestelmän tapahtumia, kuten murtohälytyksen asettamista sekä kaikkien hälytysten laukeamista, kuittaamista ja poistumista. Koska loki haetaan sivu kerrallaan tietokannasta, vieritystä ei voitu soveltaa tähän näkymään sellaisenaan, koska vieritys dynaamisella sisällönlataamisella olisi ollut liian aikaavievä ratkaisu. Lokisivujen navigointi onkin toteutettu perinteisillä sivunvaihtopainikkeilla.

Puhelinnumeronäkymässä järjestelmään voidaan rekisteröidä enintään neljä puhelinnumeroa, joihin ohjausyksikkö lähettää tekstiviestin kun esimerkiksi lämpötila-anturien lukemat laskee hälytysrajan alle tai murtohälytin laukeaa. Puhelinnumeron syöttämistä varten avataan dialogi, jossa on yksinkertainen numeronäppäimistö. Numeronäppäimistö on toteutettu erikseen, eikä käytä tekstinsyötössä käytettyä virtuaalinäppäimistöä.

Yleisissä asetuksissa voidaan vaihtaa murtohälyttimen koodi toiseen avautuvan dialogin kautta. Käyttäjän pitää ensin syöttää murtohälyttimen vanha koodi, minkä jälkeen hän syöttää uuden koodin kahteen kertaan näppäilyvirheiden ehkäisemiseksi. Koodinsyöttöön käytetään samaa numeronäppäimistöä kuin puhelinnumeroiden syötössä. Näkymässä voidaan vaihtaa myös laitteen kellonaika samanlaisen numeronäppäimistön avulla. Kellonajan vaihtaminen vaatii myös ohjauspäätteen uudelleenkäynnistyksen, jotta myös ajastukset päivittyvät oikein. Näkymässä on myös erikseen vaihtoehto päätteen uudelleenkäynnistämiselle, sekä päätteen äänenvoimak-

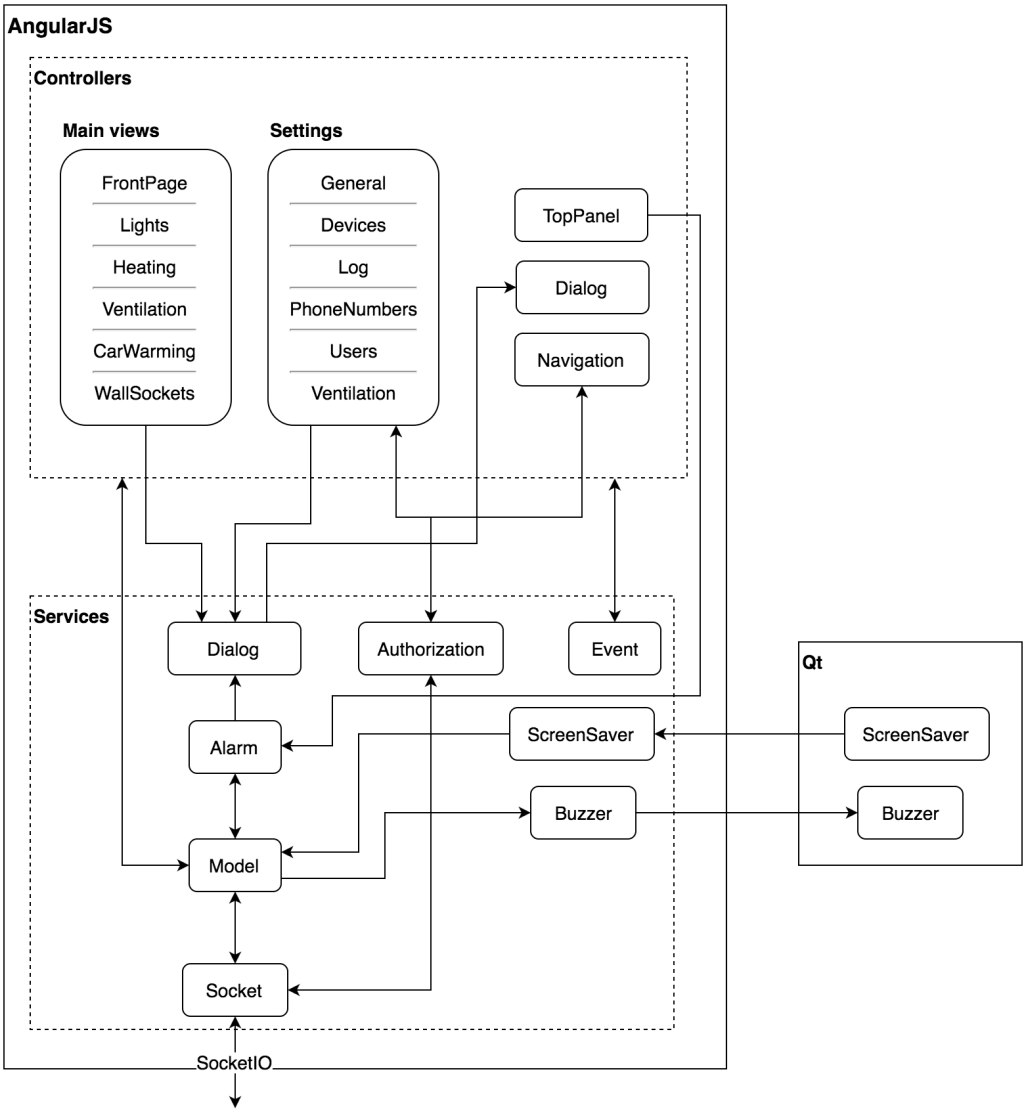
kuutta voidaan säätää liukukytkimen avulla. Äänenvoimakkuuden säätö vaikuttaa vain mm. murtohälytyksen poistumisviiveen ilmaisuun, mutta hälytystilanteissa äänimerkit soitetaan aina täydellä äänenvoimakkuudella. Lisäksi yleisissä asetuksissa ovat esitettyinä ohjelmiston versio sekä laitteen IP-osoitteet.

Kiinteiden näkymien ja edellä mainittujen dialogien lisäksi järjestelmässä esitetään dialogeja myös hälytys- ja virhetilanteissa, eli palo-, häkä- tai kosteushälytyksissä sekä esimerkiksi akun alhaisen varauksen tai katkenneen CAN-yhteyden tapauksissa. Hälytykset ja virhetilanteet voidaan kuitata päätteestä, mutta aktiiviset hälytykset jäävät näkyviin yläpaneeliin ikoneiksi. Dialogi esitetään myös, kun käyttäjä kytkee murtohälyttimen päälle, jolloin dialogi esittää numeronäppäimistön sekä näyttää jäljellä olevan poistumisajan sekunteina. Jos täysi murtohälytin on kytketty päälle, ohjauspäätettä ei voida käyttää ennen kuin dialogiin on syötetty hälyttimen oikea koodi. Kuorivalvonnan sekä lisärakennuksen murtohälyttimen ollessa päällä päätettä voidaan kuitenkin käyttää normaalisti, jolloin hälyttimen pääsee kytkemään pois päältä päätteen hälytyssivulta.

5.5 Frontend

Frontendin arkkitehtuuri on esitetty kuvassa 5.6. Ohjaimet ja palvelut ovat kaaviossa eriteltyinä, koska palvelut toimivat enemmänkin rajapintoina backendille ja Qt-ohjelmalle, sekä myös eri ohjainten välisinä viestienvälittäjinä. Kukin ohjain on liitetty johonkin näkymään, mikä tarkoittaa joko kokonaista näkymäsivua tai muuta näkymäkokonaisuutta. Yksittäisiä näkymäelementtejä ei ole arkkitehtuurikaaviossa esitettyinä, koska ne ovat enimmäkseen triviaaleja HTML-elementtejä tai AngularJS-direktiivejä, eivätkä ne ole merkityksellisiä arkkitehtuurin kannalta. Joitakin riippuvuusyhteyksiä palvelun ja useamman ohjaimen välillä on esitetty kaavion selkeyden takia yksittäisenä nuolena, jos riippuvuus koskee kaikkia ohjaimia. Ohjaimia ovat myös ryhmitetty näkyvyyden kannalta päänäkymän, asetusnäkymän ja itsenäisien näkymien ohjaimiin.

Frontend-ohjelman keskiössä on mallipalvelu (model service), joka on yhteydessä käytännössä kaikkiin ohjaimiin. Koska malli on käytännössä yksi JSON-muodossa tallennettu olio, voidaan jokaisessa ohjaimessa määritellä mitä mallin polkuja ohjain tarkkailee. Mallin polku liitetään ohjaimen AngularJS-scopeen, josta mallin sisältöä voidaan esittää suoraan ohjaimeen liitetyssä HTML-näkymässä. Mallin muuttumista voidaan myös tarkkailla ohjaimessa ja liittää muutokselle käsittelyfunktio scopen



Kuva 5.6 *Frontend-arkkitehtuurikaavio.*

watch-funktion avulla, jolloin voidaan esimerkiksi avata dialogeja hälytysten lauetessa.

Mallipalvelu synkronoi mallin socketpalvelun avulla, joka ottaa backendiin yhteyden ja päivittää frontendin mallia SocketIO-rajapinnasta tulevien muutostapahtumien perusteella. Mallin päivityksestä lähetetään tapahtuma rajapinnan kautta backendin puolelle, jossa malliin tehdään muutos. Kaikki komennot esimerkiksi CAN-viestien lähettämiseksi frontendiltä backendille ovat myös toteutettu mallipalvelujen välisessä kommunikoinnissa.

CAN-väylästä tulevia äänimerkin soittamiskäskyjä välitetään mallipalvelun kautta buzzerpalveluun, joka ohjaa Qt-ohjelman äänimerkkitoiminnallisuutta. Qt-ohjelmalla voidaan alustaa Webkit-selainohjelman käynnistyessä JavaScript-moottoriin globaaleita olioita, joihin voidaan liittää funktioiksi Qt:n signaaleja. Qt-ohjelman toiminnallisuuksia, kuten esimerkiksi äänimerkin soittamista, voidaan tällä tavalla kutsua frontend-ohjelmasta.

Näytönsäästäjän toiminnallisuus on määritelty Qt-ohjelman puolella, jossa näytönsäästäjän tilaa ylläpidetään eri tasoilla. Ensimmäisellä tasolla pääte on normaalikäytössä, toisella tasolla näytön kirkkaus himmenee, kolmannella tasolla näytöllä esitetään asiakkaan toimittama näytönsäästäjäanimaatio mustalla taustalla peittäen ohjauspäätteen käyttöliittymän, sekä lopuksi neljännellä tasolla näytön taustavalo sammutetaan kokonaan. Näytönsäästäjän tasojen ohjaus on toteutettu aikavälien avulla, ja taso palautuu aina ensimmäiselle tasolle kosketusnäyttötapahtumasta. Näytönsäästäjän palautuessa kolmannelta tai neljänneltä tasolta ohjauspäätteen käyttöliittymä palautuu etusivunäkymään sekä sulkee avoinna olevat dialogit murtohälytin- ja hälytysdialogeja lukuunottamatta. Tieto palautumisesta välitetään mallipalvelun kautta ohjaimille sekä backendin puolelle.

Murtohälytinpalvelu (alarm) tarkkailee mallipalvelun kautta aktiivisen murtohälyttimen tyyppiä. Jos tyyppi on täysi murtohälytin ja se on kytkettynä päälle, avataan aina modaalinen dialogi, jota ei saa suljettua muuten kuin kytkemällä hälyttimen pois päältä syöttämällä dialogiin oikean koodin. Kuorivalvonnan sekä lisärakennuksen valvonnan tapauksessa hälytindialogi on suljettavissa. Murtohälytindialogi avataan murtohälytinpalvelun kautta yläpaneelin ohjaimesta, koska yläpaneeliohjain on ajossa kaikissa päätteen näkymissä.

Dialogipalvelu ohjaa kaikkien erityyppisten dialogien avaamista ja niiden tulosten käsittelyä. Useista näkymistä on yhteys dialogipalveluun, ja palveluun on toteutettu jokaiselle dialogityypille oma avaamisfunktio. Palvelu välittää dialogin avauskäskyn dialogiohjaimelle, joka lopulta avaa dialogin ruudulle. Itse dialogikomponentit ovat DOM-puun juuressa. Ohjelmassa voidaan avata lisää dialogeja toisten dialogien sisältä, joten ohjelma sisältää kaikkiaan 4 komponenttia eritasoisille dialogeille. Jokaiselle dialogityypille on määritelty oma taso mm. tärkeyden perusteella – esimerkiksi hälytysdialogi näytetään aina muiden dialogien päällä. Jokaiselle dialogityypille on myös määritelty oma HTML-pohja sekä aliohjain.

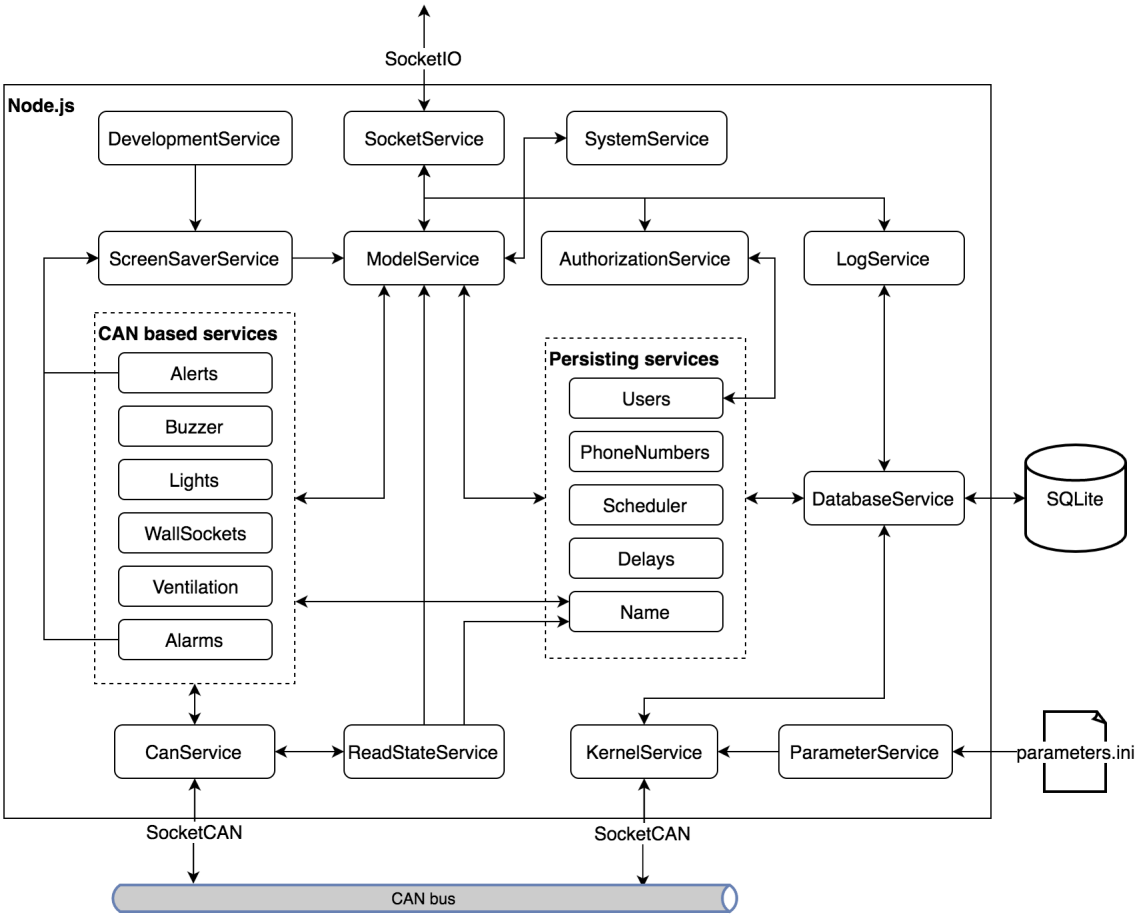
Valtuutuspalvelun (authorization) avulla käyttäjä voi tunnistautua antamalla tun-

nuksensa asetusnäkymän sisäänkirjautumisessa. Palvelu lähettää tunnukset backendiin suoraan socketpalvelun kautta, ja saa takaisin vastauksen kirjautumisen onnistumisesta. Navigointipaneeli saa kirjautumisesta myös tiedon, ja näkymässä esitetään linkit asetussivuille. Kaikkien asetusnäkymien ohjaimien toiminta on estetty sekä navigointilinkit piilotetaan, jos käyttäjä ei ole sillä hetkellä sisäänkirjautuneena. Sisäänkirjautumisessa on aikaraja, jonka jälkeen käyttäjä uloskirjataan. Kosketusnäytön käyttäminen sisäänkirjautuneena kuitenkin nollaa aikarajan. Tämä on toteutettu lähettämällä backendille väliajoin *keepalive*-viestejä, jos kosketusnäyttötapahtumia on ollut tietyn aikavälin sisällä.

Tapahtumapalvelu (event) toimii eräänlaisena viestinvälittäjäpalveluna eri ohjaimien välillä. Tapahtumat välitetään AngularJS:n juuriscopen (root scope) kautta koko ohjelmalle.

Koska joissakin näkymissä sisältö voi venyä yli ruudun rajojen eikä Qt Webkit tue vierittämistä kuin vierityspalkin avulla, ohjelmaan toteutettiin vieritysmekanismi. Vieritystä käytetään eri puolilla järjestelmää, joten se toteutettiin AngularJS-direktiivinä, jonka voi antaa attribuuttina mille tahansa HTML-elementille, joka sisältää mahdollisesti vieritettävää sisältöä. Vieritys aktivoituu, kun käyttäjä koskee mitä tahansa vieritettävän sisällön kohtaa, ja sivun sisältö alkaa seurata sormea kun se liikkuu ruudun pinnalla. Vierityksen aloittamisessa on kuitenkin pieni kynnys, koska painikkeiden painamisessa sivu voi muuten liikkua hieman. Kynnys ei kuitenkaan ole juurikaan havaittavissa. Usein älylaitteissa huomattavaa vierityksen momenttia ei voitu toteuttaa, koska Qt Webkitin vanhentuneen JavaScript-moottori ei tunnistanut funktiota, jolla saadaan laskettua vierityksen nopeus vertaamalla kehyksiä keskenään.

Kaikki käyttöliittymässä esitetyt tekstit ovat lokalisoitavissa. Lokalisointia varten kehitettiin yksinkertainen lokalisointimoduuli, mikä kääntää polkumuotoisen lokalisaatioavaimen tekstiksi. Lokalisoidut tekstit ovat keskitettynä yhteen tiedostoon, missä määritellään ohjelman kaikki tekstit yhdessä oliossa. Lokalisoinnissa voidaan myös antaa parametreja tekstin seassa esitettäviksi. Lokalisointifunktio on käytettävissä juuriscopen kautta, jolloin ei tarvitse määritellä jokaiselle sitä tarvitsevalle ohjaimelle lisäriippuvuuksia uudesta palvelusta, vaan funktio on suoraan käytettävissä HTML:ssä. Kaikki lokalisointi on tällä hetkellä vain suomeksi, mutta lokalisoinnin käyttöönottaminen alusta asti helpottaa esimerkiksi saman tekstin muokkaamista useaan näkymään sekä tuo valmiuden useamman kielen tuelle.



Kuva 5.7 *Backend-arkkitehtuurikaavio.*

Frontendin virheloki ohjataan Qt:n puolelle JavaScriptin console-olion kautta. Loki tulostetaan terminaalissa, jossa Qt-ohjelma on ajossa. Qt-ohjelman tulosteet kerätään lokitettaviksi tiedostoihin, jolloin voidaan selvittää joitain tapahtuneita virhetilanteita, kun huoltohenkilökunta kopioi päätteen lokitiedostot talteen.

5.6 Backend

Ohjelmiston backend koostuu Node.js -palvelinohjelmasta, SQLite-tietokannasta, päätteen CAN-väylärajapinnasta sekä käyttöjärjestelmän rajapinnoista. Backendin ohjelmistoarkkitehtuuri on esitetty kuvassa 5.7.

Backend on yhteydessä frontendiin socketpalvelun kautta, mikä lähettää ja vastaanottaa viestejä SocketIO-rajapinnan välityksellä. Jokaiselle socketpalvelua suoraan

käyttävälle palvelulle luodaan oma nimiavaruus, joiden avulla eri palveluita koskevat tapahtumat erotetaan toisistaan. Socketpalvelua käyttävät suoraan mallipalvelu (model service), valtuutuspalvelu (authorization service) sekä lokipalvelu (log service).

Toinen keskeinen palvelu on tietokantapalvelu. Tietokantapalvelu suorittaa tietokantakyselyitä SQLite-tietokantaan muodostamalla kyselyt tekstimuotoisina SQL-lauseina. Tietokantapalvelu tarjoaa palvelimen muita palveluita varten JavaScript-funktiot tietokannan käsittelyyn, joten SQL-kyselyitä ei tarvitse muodostaa erikseen jokaisessa sitä käyttävässä palvelussa. Erillinen tietokantapalvelu mahdollistaa myös sen, että tietokantaa vaihtaessa voidaan toteuttaa vain uusi tietokantapalvelu, jota voidaan käyttää samanlaisten funktioiden avulla, eikä muutos näin aiheuta lisätöitä muihin palveluihin. Tietokannan JavaScript-yhteensopivuuden vuoksi käytännössä kaikki tietokantapalvelulla tallennettava data sarjallistetaan JSON-merkkijonoksi. JSON-muotoinen data on helposti muutettavissa JavaScript-olioksi ilman oliorelaatiomallinnustyökaluja.

CAN-väylän yhteys on toteutettu järjestelmässä CAN-palvelulla. CAN-palvelun kautta muut palvelimen palvelut voivat lähettää taloautomaatiojärjestelmän tukemia ennaltamäärätyn muotoisia CAN-viestejä. CAN-palvelu tarjoaa muille palveluille viestin lähetysfunktion sekä mahdollisuuden kuunnella tietyn tyyppisiä viestejä callback-funktion avulla, jota kutsutaan viestin sisällöllä kun viesti on vastaanotettu. CAN-viestejä lukeviin palveluihin on konfiguroitu CAN-viestien käsittely omaan JavaScript-tiedostoonsa. JSON-muotoisen konfiguroinnin avulla saadaan eroteltua CAN-viestien käsittelylogiikka itse viestien käsittelystä kussakin palvelussa, joten niitä on nopeampi muokata jälkikäteen. Jokaiselle järjestelmän CAN-viestille on binäärisen muotonsa lisäksi englanninkielinen nimi merkkijonona, mikä helpottaa kehitystä tehden myös koodista helpommin luettavaa. Kaikkien viestien nimet ja niiden binääriset vastineet ovat myös omassa tiedostossaan.

CAN-palvelu tukee vain kehittäjien alunperin tiedossa olleita tilaa ilmaisevia sekä tilan kyselyviestejä. Esimerkiksi samaa anturia koskevissa viesteissä on sama muoto lukuun ottamatta ensimmäistä tavua, joka kertoo tyypin, onko viesti tila- vai kyselyviesti. Näiden kahden viestityypin lisäksi järjestelmässä kuitenkin ilmeni olevan ns. *kernelviestejä*. Kernelviestien avulla ohjausyksiköiden muistiin voidaan tallentaa tiettyjä arvoja (“parametreja”), kuten murtohälyttimen poistumisviiveen ja murtohälyttimen koodin. Kernelviestit poikkeavat muista CAN-viesteistä muotonsa

puolesta sen verran, että tuolloin jo toteutetusta CAN-palvelusta ei ollut suurta hyötyä. Aikataulun takia päädyttiin toteuttamaan kernelin luku- ja kirjoitusviestit kernelpalvelusta suoraan CAN-väylärajapintaa käyttämällä. Järjestelmään toteutettiin myös mahdollisuus määritellä järjestelmän parametrit erillisessä tiedostossa, jonka parametripalvelu lukee päätteen käynnistyessä. Parametrit tallennetaan ohjausyksiköiden muisteihin kernelpalvelun avulla.

Lokipalvelu käsittelee asetuksien lokinäkymässä esitettävien lokitettujen tapahtumien tallennuksen sekä haun tietokannasta tietokantapalvelun avulla. Lokipalvelulla voidaan hakea lokitapahtumia sivu kerrallaan käyttöliittymää varten. Lokipalvelun kautta mikä tahansa muu palvelinohjelman palvelu voi lisätä uuden tapahtuman lokiin. Projektin ensimmäisessä vaiheessa lokiin tallennetaan kaikki järjestelmän havaitsemat vikatilanteet sekä hälytysten laukeamiset ja kuittaamiset. Lokitapahtumaan liitetään myös lisätietoihin hälytyksen tai vian syy, jos sellainen on tiedossa ja saatavilla. Hälytyksien syyksi ilmoitetaan pääsääntöisesti hälytyksen lukeneen anturin tunniste.

Valtuutuspalvelu tarkistaa käyttäjätunnuksen ja salasanan, kun käyttäjä yrittää kirjautua asetuksiin. Tunnusten oikeellisuus tarkistetaan käyttäjäpalvelun kautta, joka puolestaan hakee tunnukset asetusmallista mallipalvelusta. Jos tunnukset annettiin oikein, käyttäjälle annetaan oikeudet ja käyttöliittymä siirtyy esittämään asetusnäkymää. Kirjautumiselle on yhden minuutin aikaraja ennen kuin käyttäjä kirjataan automaattisesti ulos, mutta jokaisesta tuona aikana tapahtuneesta käyttöliittymätapahtumasta palvelin vastaanottaa keepalive-viestin, mikä siirtää aikarajaa minuutilla eteenpäin. Toisin sanoen käyttäjä kirjataan automaattisesti ulos, kun käyttäjä ei ole käyttänyt päätettä minuuttiin.

Mallipalvelu ylläpitää sekä mallia ajan tasalla ja tarjoaa muille palveluille sekä suoraan socketpalvelulle mallin päivitysfunktiot. Mallipalvelu ylläpitää myös tietoa nykyisen mallin inkarnaatiosta. Inkarnaatio ilmaistaan aikaleimana, milloin malli on luotu. Tämän ansiosta voidaan päätellä, onko palvelinohjelma käynnistynyt uudestaan esimerkiksi kellonajan vaihdoksen myötä. Kun palvelin käynnistyy uudestaan, inkarnaatio saa uuden aikaleiman, ja kun frontend saa vastaansa mallin uudemmalla aikaleimalla, se päivittää näkymän lataamalla sivun uudelleen. Mallipalvelu välittää myös siihen kytketyille palveluille viestejä, joita lähetetään frontendin puolelta.

Järjestelmän ohjaukseen liittyvät palvelut jakautuvat kahtia palvelimen puolella: persistoiviin palveluihin eli palveluihin, jotka tallentavat tiedot tietokantaan, sekä

CAN-väylään perustuviin palveluihin, eli palveluihin, jotka perustuvat vain CAN-väyläliikenteeseen. Molemman tyyppiset palvelut päivittävät suoraan mallia mallipalvelun kautta ja tarjoavat edelleen muille palveluille turvallisen rajapinnan palveluja koskevia toimintoja varten.

Käyttäjäpalvelu on persistoiva palvelu, joka tarjoaa tunnusten tarkistamisen lisäksi funktioita uusien käyttäjien luomista ja poistamista sekä salasanan vaihtamista varten. Lisäksi se sisältää validointifunktiot, joilla varmistetaan, että uuden käyttäjän tunnukset ovat valideja, eivätkä sisällä esimerkiksi laittomia merkkejä tai ole liian pitkiä. Salasanojen tiivisteiden luomiseen ja tarkistamiseen käyttäjäpalvelu käyttää *Scrypt*-kirjastoa. Lisäksi käyttäjäpalvelu pitää kirjaa siitä, mitkä ovat käyttäjien oikeustasot. 2-tason käyttäjät voivat vaihtaa vain omaa salasanaansa, 1-tason käyttäjät voivat lisätä ja poistaa muita käyttäjiä, ja 0-tason käyttäjät voivat edellä mainittujen lisäksi myös nollata ylläpitotunnukselle oletussalasanan. Ylläpitotunnusta ei voi kuitenkaan kukaan käyttäjä poistaa. Lisäksi huoltohenkilökuntaa varten käyttäjäpalveluun on määritelty tunnukset, joiden avulla mihin tahansa laitteeseen pääsee kirjautumaan huoltotoimenpiteitä varten.

Puhelinnumeropalvelu tallentaa asetuksissa asetettavat puhelinnumerot tietokantaan tai poistaa niitä. Puhelinnumeroita voidaan tallentaa CAN-viestien rajoituksista johtuen neljä sarjaa kerrallaan. Jos puhelinnumero lisätään, se päivitetään myös ohjauspäätteille CAN-viesteillä. Käytössä olevan CAN-viestiformaatin takia yhdellä CAN-viestillä voidaan kuitenkin välittää viestin tunnisteen lisäksi dataa yhden tavun kerrallaan, joten jokainen numero lähetetään omana viestinään. Tämä aiheuttaa hieman ylimääräistä viestiliikennettä, mutta puhelinnumeroiden päivitystä tapahtuu niin harvoin, ettei sillä ole käytännössä mitään merkitystä järjestelmän toiminnan kannalta. Viesteissä kerrotaan myös puhelinnumeron indeksi, minkä avulla on mahdollista esimerkiksi ylikirjoittaa jokin puhelinnumero toisella.

Järjestelmän viikkoajastukset toimivat skedulointipalvelun avulla. Kun käyttäjä lisää jollekin laitteelleen yksittäisen viikkoajastussäännön, se tallennetaan tietokantaan, jotta ajastus säilyy myös ohjelmiston tai laitteen uudelleenkäynnistyksessä. Yksi viikkoajastussääntö koostuu viikonpäivistä sekä alku- ja loppukellonajasta, jolloin laitteen halutaan olevan päällä. Laitteelle voidaan määritellä useita sääntöjä, mutta backend vastaanottaa lisätyt säännöt yksitellen. Lisäyksessä tarkistetaan, että aikaväli on validi ja risteäviä ajastuksia ei ole asetettu laitteelle – esimerkiksi jonkun säännön loppuaika ei saa olla ennen toisen säännön alkuaikaa, jos säännöt koskevat

samaa viikonpäivää. Myös kellonajat sekä viikonpäivät validoidaan, ettei ajastusta käynnistetä virheellisillä arvoilla. Kun ajastus on lisätty, se lisätään *node-scheduler* -kirjaston avulla ajamaan alku- ja loppuajankohtina käynnistys- ja sammutusfunktiot.

Viiveajastuksia varten palvelimella on viivepalvelu. Viivepalvelu ottaa vastaan mallipalvelusta käskyn, kun käyttäjä lisää laitteelle viiveajastuksen käynnistymään tietyn ajan päästä ja pysymään päällä tietyn aikaa. Käskyn mukana frontend lähettää viivepalvelulle ajastuksen alku- ja loppukellonaika. Näistä kellonajoista päätellään, minkä ajan kuluttua laite käynnistetään ja sammutetaan. Nykyhetken ja kellonajan erotus lasketaan millisekunteina ja laitteen käynnistys- ja sammutusfunktiot asetetaan ajettaviksi JavaScriptin *setTimeout*-funktion avulla. Timeout- ja intervallifunktioita ei kuitenkaan käytetä palvelimella suoraan, vaan niihin on lisätty omaa toiminnallisuutta, minkä avulla ajastetut funktiokutsut ovat globaalisti keskeytettäviä. Keskeytystä tarvitaan esimerkiksi kun järjestelmä sammutetaan. Kun palvelinohjelmisto käynnistyy, ladataan mahdollisesti keskeytyneet viiveajastukset tietokannasta, ja niille asetetaan uudestaan timeoutit.

Nimipalvelun avulla ylläpidetään eri laitteiden nimiä, joita käyttäjä asettaa asetusnäkymässä. Laitteille ja antureille on määritelty oletusnimet lokalisointiavaimilla, joiden avulla oikea teksti haetaan frontendin puolella. Joidenkin nimien, kuten esimerkiksi murtohälytystyyppien *täysi* ja *kuori*, muuttaminen on kuitenkin estetty sekaannusten ja mahdollisen väärinkäytön välttämiseksi. Lisärakennuksen murtohälytyksen nimi voidaan kuitenkin muokata tarpeen mukaan esimerkiksi autotalliksi. Laitteiden oletusnimimäärittelyt ovat toteutettu kunkin vastaavan palvelun konfigurointitiedostoon, jotta koodi pysyy järjestyksessä aihealueiden mukaan. Nimet tallennetaan myös tietokantaan, jotta ne eivät katoa palvelimen käynnistyessä uudelleen.

Järjestelmän hälytykset muodostetaan vastaanotetuista CAN-viesteistä hälytyspalvelussa. Hälytyksiä järjestelmässä on eri tasoisia, ja jotkut hälytyksistä esitetään frontendiin avautuvana ikkunana, kun taas jotkut esitetään vain yläpaneelissa, kuten esimerkiksi avoimien ovien lukumäärä. Murtohälytystä lukuunottamatta hälytykset voidaan kuitata, mutta aktiivista hälytystä indikoidaan käyttöliittymän yläpaneelissa. Lisäksi hälytyspalvelu käsittelee kaikki järjestelmävirheviestit, joista esitetään käyttöliittymässä kertaluontoinen kuittausikkuna. Virheet tallennetaan myös lokiin mahdollista myöhempää tarkastelua varten. Tällaisia järjestelmävirheitä ovat

esimerkiksi CAN-väyläyhteyden katkeaminen tai ohjausyksikön katoaminen. Hälytysten tilat päivitetään frontendille mallipalvelun kautta, ja niiden ilmestyminen myös herättää ohjauspäätteen näytönsäästäjän

Palvelimen buzzerpalvelu kuuntelee CAN-väylän viestejä, ja jos jokin äänimerkin käskevistä viesteistä havaitaan, palvelu lähettää äänimerkkikäskyn frontendille mallipalvelun kautta. Vastaavasti jos ohjauspäätteen frontend- tai palvelinohjelmissa käytetään äänimerkkejä, ne välitetään mallipalvelun käskyn avulla CAN-väylään, jolloin mahdollisesti muut väylään kytketyt päätteet soittavat myös äänimerkin. Suurin osa äänimerkkikäskyistä ovat peräisin ohjausyksiköistä, mutta esimerkiksi murtohälytintä kytkettäessä päälle poistumisviivettä ilmaistaan sekunnin välein kuuluvilla äänimerkeillä.

Valopalvelu ylläpitää järjestelmän valaistuksen tilaa. Palvelimen käynnistyessä valopalvelu lähettää kyselyviestien sarjan CAN-väylään valaistusryhmien ja valaistustilojen selvittämiseksi. Kun palvelu saa paluuviestinä eri valaistusten tilat, saadaan selville mitkä valaistusryhmät ja -tilat ovat asennettu järjestelmään, ja siten voidaan esittää valaistuksille oikeat kytkimet käyttöliittymässä. Binäärisen päälle- ja poiskytkennän lisäksi joidenkin valaistusryhmien kirkkautta voidaan säätää. Tähän on olemassa oma viestinsä, mutta ohjausyksiköt eivät tallenna kirkkautta muistiin, joten kirkkaus palautuu takaisin kun valaisin kytketään pois päältä ja uudestaan päälle. Kirkkauden tallentamiseen ainoa vaihtoehto on simuloida seinäkytkinten toimintaa eli painikkeen pohjassa pitämistä kunnes haluttu kirkkaus saavutetaan. Tämä on toteutettu lähettämällä CAN-väylään viestit painikkeen alas painamisesta ja irti päästämisestä. Palvelu laskee kirkkausarvosta arvioidun ajan, kauanko painiketta pidetään pohjassa.

Valopalvelussa on lisäksi rajapinta valaistustilan uudelleenohjelmoinnin aloittamiseen ja lopettamiseen. Uudelleenohjelmoinnin aikana käyttäjä valitsee mitkä valot ovat päällä ja millä kirkkaudella. Kun uudelleenohjelmointi lopetetaan, lähetetään ohjausyksiköille viesti tilan uudelleenohjelmoinnin hyväksymisestä. Ohjelmointi voidaan myös peruuttaa ottamatta muutoksia voimaan.

Pistokepalvelu ylläpitää pistokkeiden, lämmityksen sekä autonlämmittimien tiloja, koska ne ovat toiminnallisuudeltaan samankaltaisia. Valopalvelun tapaan pistokepalvelu kysyy palvelimen käynnistyessä laitteiden nykyistä tilaa. Lisäksi palvelu käynnistää laitteiden ajastukset, jos skedulointi- tai viivepalvelusta löytyy laitteille aktiivisia ajastuksia.

Ilmanvaihtopalvelun avulla ohjataan ilmastoinnin tehostusta ja sen rajojen asettamista. Ilmastoinnin tehostuksen rajoiksi määritellään ulkolämpötila- ja ilmankosteusrajat sekä viive, minkä kuluttua tehostus otetaan käyttöön rajaehtojen täyttyessä. Lämpötilaraja muunnetaan CAN-väylää varten kahden komplementtiluvuksi. Ilmastoinnin tehostuksen lisäksi lämpötila-anturien poikkeamien asettaminen on ilmanvaihtopalvelun vastuulla, koska asetusnäkymässä toiminnot ovat samalla sivulla.

Murtohälytinpalvelun (alarms service) vastuulla on kaikki murtohälyttimeen liittyvä toiminta, kuten CAN-väylän murtohälytinviestien käsittely, murtohälyttimen päällekytkentä poistumisviiveineen sekä murtohälyttimen koodin tarkastaminen. Palvelu päivittää mallia CAN-viestien mukaisesti, jotta useampi samaan väylään kytketty ohjauspääte pysyisi keskenään synkronisoituina. Jos murtohälytin on päällä ja liiketunnistin havaitsee liikettä, palvelu herättää ohjauspääteruudun näytönsäästäjästä ja antaa käyttäjälle parametreissä määritellyn aikarajan syöttää koodi ennen kuin hälytys laukeaa. Jos käyttäjä syöttää väärän koodin liian monta kertaa, ohjauspäätteen käyttö lukitaan joksikin aikaa. Hälyttimen koodin vaihto on toteutettu kolmivaiheisena – käyttäjä syöttää ensin nykyisen koodin, jonka jälkeen uusi koodi kysytään kahteen kertaan näppäilyvirheiden välttämiseksi. Koodin vaihto peruutetaan, jos jokin vaiheista epäonnistuu.

Tilanlukupalvelua käytetään lukemaan niitä järjestelmän laitteita tai antureita, mille ei ole omaa palveluaan. Tämä käsittää järjestelmän lämpötila- ja kosteusanturit sekä akun jännitteen. Palvelimen käynnistyessä palvelu rekisteröi antureiden oletusnimet nimipalveluun. Lämpötilojen lukemista varten palveluun toteutettiin myös muunnosfunktio, jolla lämpötila voidaan muuntaa CAN-viesteissä välitetystä kahden komplementtimuodosta desimaaliseen muotoon. Akun jännitteelle on oma heksadesimaalinen esitysmuotonsa CAN-viesteissä erikoisemman vaihteluvälin takia. Tämä esitysmuoto takaisinmallinnettiin ja sille toteutettiin myös oma muunnosfunktionsa.

Järjestelmäpalvelu tarjoaa rajapinnan ohjauspäätteen järjestelmätason toiminnoille. Järjestelmän aika vaihdetaan tässä palvelussa, jolloin myös palvelinprosessi käynnistetään uudelleen, jotta ajastukset ottaisivat huomioon muuttuneen kellonajan. Kellonajan päivityksestä lähetetään frontendille päivityskäsky minuutin välein, jolloin palvelimen kello ja frontendin kello pysyvät aina samassa ajassa. Tämä on tärkeää varsinkin silloin, kun frontendiä suoritetaan mobiililaitteella tai tulevan pilvipalvelun kautta, jolloin selainlaitteen kello voi olla merkittävästikin eri ajassa palvelimeen

verrattuna.. Koko ohjauspääte voidaan myös käynnistää uudelleen palvelun avulla. Lisäksi ohjelmiston versiotiedot sekä laitteen IP-osoitteet ovat myös saatavilla tämän palvelun kautta.

Lisäksi kehitystä varten backend-ohjelmistossa on kehityspalvelu, joka valvoo frontendin kooditiedostoissa tapahtuvia muutoksia. Jos muutos havaitaan, näytönsäästäjä herätetään sekä sivu ladataan uudestaan. Tämä on tärkeä toiminnallisuus varsinkin oikeaa ohjauspäätettä käytettäessä, koska sivun lataaminen on päätteellä paljon monimutkaisempaa tavalliseen selaimeen verrattuna.

Koko palvelinohjelmiston alustus on keskitetty moduulilataajaan (module loader). Kaikki palvelut tarjoavat alustusfunktion, joka ajetaan moduulilataajan toimesta palvelimen käynnistyessä. Ohjelman debug- ja virhelokitukseen käytetään *winston*-kirjastoa, jossa on toteutettuna valmiiksi lokirotaatio, ja myös eri tasoiset lokiviestit voidaan erotella toisistaan. Viestien ja nimien lokalisointia varten kaikki tekstit lähetetään frontendille lokalisointiavaimina, ellei nimi ole käyttäjän määrittelemä. Tämän ansiosta kaikki lokalisaatio on keskitettynä frontendin puolelle yhteen tiedostoon.

6. ARVIOINTI

Tässä luvussa käydään läpi, kuinka hyvin laajennus saavutti tavoitteet. Lisäksi luvussa käydään läpi työhön liittyviä ongelmia ja haasteita, asiakkaalta vastaanotettua palautetta sekä kokemuksia kehityksestä. Lopuksi esitellään laajennukselle mahdollisia jatkokehityskohteita.

6.1 Tavoitteiden toteutuminen

Tavoitteille ei asetettu projektissa numeerisesti mitattavia arvoja, vaan tavoitteiden toteutumisen tärkein mittari on se, kuinka tyytyväinen asiakas on projektin tavoitteiden toteutumiseen.

Tavoite ohjauspäätteen käyttöliittymälle olla edeltäjiään nopeampi toteutui selvästi, sillä päätteen ruutu reagoi käyttäjän toimenpiteisiin miltei välittömästi. Edellisen kosketusnäyttöpäätteen ruudun päivitys saattoi kestää 1-2 sekuntia, mutta uudessa ohjauspäätteessä käyttöliittymätapahtumista annetaan palaute melkein huomaamattoman lyhyessä ajassa. Joissakin isommissa näkymissä, kuten valaistusryhmien listauksessa tai dialogin esittämisessä kuitenkin vasteaika kasvaa jonkin verran. Palaute tulee aina kuitenkin tarpeeksi nopeasti pysyäkseen miellyttävänä käyttää. Nopeutta olisi voinut saada tehostettua valitsemalla frontend-teknologiaksi jonkun muun kuin AngularJS:n. Muun teknologian nopeammasta toimivuudesta ei ole kuitenkaan takeita, koska Qt Webkitin käyttämän JavaScript-moottorin suorituskyky voi olla pullonkaulana.

Ohjauspäätteen helppokäyttöisyyden mahdollistaa edellisiä ohjauspäätteitä huomattavasti suurempi näyttö. Näytön koon ansiosta käyttöliittymässä on tilaa käyttää käyttöliittymäkomponentteja, joihin kuka tahansa käyttäjä voi osua hyvällä tarkkuudella. Näytöllä on myös edellisiin verrattuna paljon enemmän tilaa esittää järjestelmän tilaa. Käyttöliittymäsuunnittelussa noudatettiin myös modernien kosketusnäyttökäyttöliittymien suunnitteluperiaatteisa, kuten esimerkiksi sivun vierittä-

minen sormella pyyhkäisemällä vierityspainikkeiden sijaan. Tämän ansiosta muihin kosketusnäyttölaitteisiin tottuneet käyttäjät osaavat käyttää käyttöliittymää sulavasti, ja tottumattomillekin käyttäjille ohjaus on nopeasti opeteltavissa.

Reaaliaikaisuuden toteutuminen voitiin todentaa kehitysvaiheessa. CAN-väylämonitoria seuraamalla sekä ohjauspäätettä käyttämällä voitiin tarkkailla, kuinka nopeasti CAN-väylän viestit käsitellään ja käyttöliittymä päivitetään. Huomattavaa viivettä ei ollut, joten reaaliaikaisuus riippuu siitä, kuinka nopeasti ohjausyksiköt lähettävät CAN-viestejä järjestelmän tilassa havaituista muutoksista. Ohjauspäätteen osalta siis reaaliaikaisuuden tavoite on toteutunut.

Web-teknologioita käyttämällä voitiin saavuttaa asiakkaan toiveiden mukainen ulkoasu käyttöliittymälle. Käyttöliittymän tyylimäärittelyissä käytettiin asiakkaan graafisen ohjeen mukaisia arvoja. Kehityksen aikana tehtiin myös muutoksia ulkoasuun esimerkiksi logon päivittämiseksi ja graafisessa ohjeessa määriteltyjen taustakuvioiden käyttämiseksi. Ulkoasun tavoite saatiin näin toteutumaan.

Kehityksen alusta asti on keskitytty siihen, että modulaarisuus toteutuu mahdollisimman hyvin. Frontendin puolella tämä tapahtuu lähinnä AngularJS-direktiivien ja -palveluiden käytöllä. Backendin puolella modulaarisuus toteutuu palvelujaon sekä palveluiden CAN-viestien konfiguroinnin erottamisella omiin tiedostoihin. Modulaarisuus näkyi kehityksen aikana erityisesti frontendin puolella komponenttien uudelleenkäytettävyytenä. Backendin modulaarisuus helpottaa varsinkin jatkokehitystä palveluiden CAN-konfigurointitiedostojen avulla. Modulaarisuuden voidaan siis katsoa toteutuneen hyvin.

Mahdollisen pilvipalvelun kehittämistä varten ohjauspääte tarjoaa palvelinrajapinnan, jota voidaan käyttää pohjana pilvipalvelun rajapinnalle. Liikennettä rajapinnan kanssa ei kuitenkaan voida sallia suoraan julkisesta verkosta tietoturvasyistä, vaan kommunikoinnin pilvipalvelimen ja ohjauspäätteen tulisi tapahtua jonkinlaisen välityspalvelimen avulla. Lisäksi pilvipalvelun käyttämää rajapintaa tulee rajoittaa, koska esimerkiksi murtohälytyksen poiskytkentää ei ole tarpeen ohjata pilvipalvelun kautta, ja se voisi muutenkin aiheuttaa vakavia turvallisuusriskejä tietomurron seurauksena tai tunnusten vuotaessa ulkopuoliselle. Käyttöliittymän lähdekoodi olisi pilvipalvelun käytettävissä pienillä muutoksilla, kuten virtuaalinäppäimistön ja kosketusnäyttövierityksen poistamalla. Valmiudet pilvipalvelun kehittämiseen jatkokehityksessä ovat kuitenkin olemassa, joten tavoite on toteutunut.

6.2 Ongelmat ja haasteet

Suurin yksittäinen haaste laajennuksen kehittämisessä oli se, että CAN-viestien formaatista ja toiminnasta oli olemassa dokumentaatiota vain osasta viestejä. Tämän takia monen CAN-viestin toiminnallisuutta jouduttiin selvittämään takaisinmallinnuksen (reverse engineering) avulla. Esimerkiksi viesti, jolla voidaan kysyä valaisinryhmien tilaa, oli tuntematon projektin loppupuolelle asti, eikä aluksi ollut edes varmuutta tällaisen viestin olemassaolosta. Viestit selvitettiin tarkkailemalla CAN-väylämonitoria samaan aikaan kun vanhoja ohjauspäätteitä ja muita laitteita käytettiin, sekä kokeilemalla lähettää valistuneisiin arvauksiin perustuneita viestejä ja tutkimalla niiden vaikutusta. Käytössä oli lisäksi ohjausyksiköiden ohjauslogiikka tilakoneina esitettynä, millä voitiin selvittää miten tiettyjen toimintojen tulisi tarkalleen toimia kun CAN-viesti vastaanotetaan ja millaisia CAN-viestejä mahdollisesti voidaan ohjausyksiköstä lähettää.

Käyttöliittymässä on käytössä hyvin vähän animaatioita sekä joitakin CSS-ominaisuuksia, koska käyttöliittymä ei toiminut kovin sulavasti niiden kanssa. Animaatiot voivat oikein käytettyinä tuoda käyttäjälle responsiivisuutta, mutta pätkivät animaatiot antavat käyttöliittymästä hitaan vaikutelman. Lisäksi esimerkiksi CSS-varjostusten käytön huomattiin hidastavan käyttöliittymää merkittävästi varsinkin tilanteessa, jossa usea dialogi varjostuksineen esitettiin ruudulla samaanaikaan – kosketustapahtuman käsittelyyn saattoi kulua jopa useampi sekunti. Nämä ongelmat johtuvat todennäköisesti vanhentuneesta Qt Webkitistä.

Projektinhallinnassa pyrittiin jonkin tasoiseen ketteryyteen pitämällä asiakkaan kanssa palavereita toteutetuista toiminnallisuuksista. Asiakkaan ajanpuutteen vuoksi palavereita kuitenkaan ei järjestetty kovin tiiviiseen tahtiin, minkä takia moni toiminnallisuus ehti valmistua palaverien välissä, ja palavereista tuli tietynlaisia esittelytilaisuuksia. Selkeisiin toimintojen ongelmakohtiin puututtiin, mutta muuten esitellyt ratkaisut saivat hiljaisen hyväksynnän. Tiukemmalla ja säännöllisellä palaveriaikataululla asiakkaalta olisi voinut saada paremmin haasteita ja pohdintaa yksityiskohtaisemmalla tasolla. Palaverit järjestettiin lisäksi aina paikan päällä, mutta asiakkaan ajan säästämiseksi muiden kiireiden vuoksi etäpalavereilla ja toimintojen esittelyllä ruudunjaon kautta olisi voinut sopia prosessiin paremmin. Projektissa ei ollut myöskään mukana varsinaista tuoteomistajaa, joka olisi ottanut tehtäväkseen varmistaa toteutuksen ratkaisut yhteistyössä kehittäjien kanssa. Projektin aikana myös sen laajuus vaihteli loppuun asti, eikä projektille ollut olemassa selkeää

määrittelyä, mitkä ovat mittarit sille, että projektin voidaan katsoa päättyneen. Lisäominaisuuksia ja muutoksia toteutukseen tuli asiakkaalta vielä projektin viime hetkillä, kunnes asiakkaan kanssa sovittiin erikseen projektin vaiheen päättyvän toimitukseen ja että lisämuutokset siirtyvät seuraavaan vaiheeseen virheenkorjauksia lukuunottamatta. Vaatimusten ja ominaisuuksien muuttuminen sekä lisääntyminen venyttivät projektin aikataulua jonkin verran.

Projektin alussa eräs haaste oli perehtyminen. Vaikka kehittäjillä oli kokemusta web-kehityksestä, käytetyistä Node.js- ja AngularJS-teknologioista ei ollut kovin laajaa kokemusta. Perehtyminen vei jonkin verran aikaa, mutta tätä helpotti kuitenkin se, että yrityksen sisäisen konsultoinnin avulla projektille saatiin frontend-ohjelman pohja sekä kehitystyökalujen käyttöönotto. Kokemuksen puute teknologioista näkyy frontendin puolella siinä, että ohjelman suorituskykyä oltaisiin voitu tehostaa, jotta ylimääräiseltä tilan tarkkailulta vältyttäisiin. AngularJS tarjoaa tilan tarkkailuun *scope.$watch*-funktion, johon toteutetun frontend-ohjelman toiminta perustuu – sen liiallinen käyttö voi kuitenkin vaikuttaa käyttöliittymän suorituskykyyn merkittävästi. Watch-funktion käyttöä olisi voinut rajoittaa suunnittelemalla tiedon kulun eri tavalla, mutta sen käyttöön päädyttiin puuttuvan kokemuksen ja käytön helppouden takia.

6.3 Kehityskokemukset

Projektin kehitys oli haastavaa mutta mielenkiintoista, koska vastaavanlaisen kosketusnäyttöohjelman kehityksestä ei ole aikaisempaa kokemusta. Myös poikkeuksellinen ympäristö ja vuorovaikutuksen kehittäminen muun taloautomaatiojärjestelmän kanssa teki projektista mielenkiintoisen, mutta se aiheutti myös paljon töitä takaisinmallinnuksen muodossa. Kehitysympäristöjen rakentaminen vei oman aikansa, mutta kun ympäristö saatiin valmiiksi, kehitys sujui ilman suurempia ongelmia.

Vaikka kehityksessä olikin käytössä kaksi hieman erilaista testiympäristöä, ne eivät välttämättä vastanneet todellista ympäristöä. Tämä johtui osin siitä, että testiympäristöissä käytettiin hieman vanhempia ohjausyksikön versioita. Ympäristöjen eroavuus tuli todennettua esimerkiksi yhdellä demotilaisuudella, joka pidettiin asiakkaan toimistolla, johon on asennettu kyseinen taloautomaatiojärjestelmä ohjaamaan valaistusta ja murtohälytintä. Kun uusi ohjauspääte liitettiin CAN-väylään ja käynnistettiin, toimiston valot sammuivat itsestään. Hetken tutkimisen jälkeen

huomattiin, että valaistusryhmien tilakysely voi sammuttaa valot, mikä ei kuitenkaan toistunut testiympäristöllä. Tämä aiheutti jonkin verran epävarmuutta siihen, kuinka hyvin ohjelmisto tukee testiympäristöjen lisäksi todellisia ympäristöjä, kun se viedään tuotantoon. Asiakkaalle toimitettiin kuitenkin projektin loppupuolella ohjauspäätteen julkaisuehdokasversioita testattavaksi oikeissa ympäristössä. Järjestelmän CAN-viestien puutteellisen määrittelyn vuoksi projekti vaati myös paljon takaisinmallinnusta – tilankyselyviestit ja niin sanotut "JUCE heart beat" -viestit pääteltiin kehityksessä itse viestiliikennettä seuraamalla sekä samalla kokeilemalla erilaisia CAN-viestejä ja tarkkailemalla niiden vaikutusta.

Yksi rajoittavimmista teknisistä asioista oli se, ettei päätteellä voitu käyttää uusinta versiota Qt:sta. Tämän takia käyttöliittymän kehityksessä ei esimerkiksi voitu käyttää kaikkia uusimpia CSS3-tyylimäärittelyjä, eikä esimerkiksi monista kosketusnäyttölaitteista tuttua vierityksen momenttia pystytty toteuttamaan, koska JavaScript-moottorilla ei ole toteutettu sen vaatimaa *window.requestAnimationFrame*-funktiota. Vanhan Qt Webkit -version JavaScript-moottorin suorituskyky on myös todennäköisesti koko frontend-ohjelman suorituskyvyn pullonkaula.

6.4 Jatkokehitysideoita

Frontend-ohjelmaan on toteutettu lokalisoinnin tuki, mutta toistaiseksi vain suomen kieli on tuettuna. Asiakaskunnan laajentamista edesauttaisi ainakin englannin kielen lisääminen tuettuihin kieliin jatkokehityksessä. Lisäksi asetusnäkymästä pitäisi voida vaihtaa kieltä mihin tahansa tuettuun kieleen ilman järjestelmän uudelleenkäynnistämistä.

Ohjauspäätteen alustalla on paikka WLAN-sirulle, mutta kehitykselle toimitetuissa päätteissä sirua ei ole asennettuna. Jos päätteisiin asennettaisiin WLAN-sirut, huoltotoimenpiteet helpottuisivat, koska ohjauspäätteeseen voisi päästä ottamaan langattomasti SSH-yhteyden. Lisäksi kantaman sisällä olevilla älylaitteilla voisi päästä ohjaamaan järjestelmää. WLAN-verkon suojaukseen tulee kuitenkin kiinnittää erityistä huomiota, ettei järjestelmään pääsisi tunkeutumaan talon ulkopuolelta. WLAN-yhteyden kautta kannattaisi estää kuitenkin varmuuden vuoksi joitakin toimintoja, kuten murtohälyttimen ohjaamista.

Jatkokehitykseen suunniteltu pilvipalvelu aiheuttaa useita jatkokehitysideoita pro-

jektille. Nykyinen frontend-ohjelma on ajettavissa sellaisenaan tabletin ruudulta, mutta todennäköisesti käyttötapaukset keskittyisi älypuhelimiin niiden yleisyyden vuoksi. Tätä varten käyttöliittymä voitaisiin suunnitella uudelleen älypuhelinten näytölle sopivaksi. Tämä tulisi toteuttaa kuitenkin käyttäen responsiivista suunnittelua, jotta erillistä puhelinnäkymää ei tarvitsisi ylläpitää. Web-ohjelma voitaisiin myös kääntää älypuhelinohjelmaksi esimerkiksi Apache Cordovan avulla, jolloin ei tarvitsisi avata selainta pilvipalvelun käyttöä varten.

7. YHTEENVETO

Tämän työn tarkoitus oli kehittää valmiiseen taloautomaatiojärjestelmään laajennuksena kosketusnäyttöpäätteelle ohjelmisto, jolla esitetään järjestelmän ohjauksen käyttöliittymä sekä johon toteutetaan järjestelmään liittyviä lisäominaisuuksia. Järjestelmän laitteet kytkeytyvät keskenään CAN-väylän avulla, ja laitteiden ohjauslogiikka on keskitetty ohjausyksiköihin, jotka ohjaavat laitteita vastaanotettujen CAN-viestien mukaisesti. Laajennuksen ohjelmistoa suoritetaan uusilla kosketusnäyttöpäätteillä, joissa on valmiina liitännät CAN-väylää varten. Laajennuksen ensisijainen tavoite oli kehittää nopeampi ja käytettävämpi ohjauspääte järjestelmään.

Ohjelmisto päädyttiin toteuttamaan web-teknologioilla, koska silloin ohjelmisto tarjoaisi palvelinrajapinnan mahdollisesti jatkokehityksessä toteutettavaa pilvipalvelua varten, sekä myös käyttöliittymäkoodi olisi uudelleenkäytettävää. Käyttöliittymän esittäminen päätettiin toteuttaa Qt-ohjelmana, jossa web-sivu avataan Qt Webkit -näkymään. Arkkitehtuurin suunnittelun mallina käytettiin MVC-mallia, jota myös valittu frontend-teknologia tukee. Lisäksi ohjelmisto jaettiin toimintokohtaisiin palveluihin, jotka edesauttavat ohjelmiston modularisuutta.

Laajennuksen tavoitteet täyttyivät selvästi ja asiakas oli tyytyväinen lopputulokseen. Toimitus kuitenkin viivästyi aikataulusta, koska projektin laajuus vaihteli kehityksen viime metreille asti. Lisäksi kehityksen alussa järjestelmään sekä teknologioihin perehtyminen vei aikaa.

Vaikka käyttöliittymä onkin huomattavasti edellisiä ohjauspäätteitä nopeampi, niin sitäkin voisi vielä nopeuttaa vaihtamalla uudempaan Qt-versioon. Uusi versio tukisi myös paremmin uusia CSS-ominaisuuksia.

LÄHTEET

[1] S. Allamaraju, K. Avedal, R. Browett, J. Diamond, and J. Griffin, *Java server programming*, 2001, vol. I.

[2] P. Baronti, P. Pillai, V. W. Chook, S. Chessa, A. Gotta, and Y. F. Hu, "Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards," *Computer Communications*, vol. 30, no. 7, pp. 1655—-1695, May 2007.

[3] A. Berson, *Client/server architecture*, 2nd ed. McGraw-Hill, 1996.

[4] R. Branas, *AngularJS Essentials.* Packt Publishing, 2014.

[5] E. DeBill, "Module counts," [WWW], http://www.modulecounts.com/. [Haettu 1.11.2016.].

[6] T. Enwall and M. Soucie, *A letter from revolv's founders*, Revolv Inc., 2016, [WWW], https://web.archive.org/web/20160301002424/http://revolv.com/. [Haettu 18.12.2016].

[7] W. W. Gibbs, "As we may live," *Scientific American*, Nov. 2000.

[8] K. Gill, S.-H. Yang, F. Yao, and X. Lu, "A zigbee-based home automation system," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 2, pp. 422–430, May 2009.

[9] D. Goodman, *Dynamic HTML: The Definitive Reference: A Comprehensive Resource for HTML, CSS, DOM & JavaScript.* O'Reilly Media, Inc., 2002.

[10] R. Harper, *Inside the Smart Home.* Springer-Verlag London, 2003, 264 p. ISBN 978-1-85233-854-1.

[11] A. Hasibuan, M. Mustadi, I. E. Y. Syamsuddin, and I. M. A. Rosidi, "Design and implementation of modular home automation based on wireless network, rest api, and websocket," *2015 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, pp. 362–367, Nov. 2015.

[12] Koninklijke Philips Electronics N.V., "How hue works – overview," 2016, [WWW], https://www.developers.meethue.com/documentation/how-hue-works [Haettu 20.12.2016].

[13] A. Leff and J. T. Rayfield, *Web-Application Development Using the Model-NiewlController Design Pattern.* IBM T. J. Watson Research Center, 2001.

[14] Linnovate, *Technologies – The MEAN Stack*, 2016, [WWW], http://learn.mean.io/#mean-stack-technologies-the-mean-stack. [Haettu 20.12.2016].

[15] T. Maynard, *Getting Started with Gulp.* Packt Publishing Ltd, 2015.

[16] Philips Semiconductors, "The i2c-bus specification," *Philips Semiconductors*, vol. 9397, no. 750, 2000.

[17] Robert Bosch GmbH, *CAN Specification – Version 2.0*, 1991, [WWW], http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf. [Haettu 21.12.2016].

[18] The Qt Company, *Qt 5.0 Changes*, 2015, [WWW], https://wiki.qt.io/Qt_5.0_Changes. [Haettu 20.12.2016].

[19] The Qt Company, *Qt Documentation – Qt Style Sheets*, 2016, [WWW], http://doc.qt.io/qt-4.8/stylesheet.html. [Haettu 20.12.2016].

[20] The Qt Company, *Qt Reference Documentation – Basic Qt Architecture*, 2016, [WWW], https://doc.qt.io/archives/qt-4.7/qt-basic-concepts.html. [Haettu 20.12.2016].

[21] The Qt Company, *Qt Reference Documentation – QtWebKit Guide*, 2016, [WWW], http://doc.qt.io/qt-4.8/qtwebkit-guide.html. [Haettu 21.12.2016].

[22] S. Tilkov and S. Vinoski, "Node.js: Using javascript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, Nov. 2010.